
UNIVERSIDAD CARLOS III DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

INGENIERÍA TÉCNICA DE TELECOMUNICACIÓN:
TELEMÁTICA

PROYECTO FIN DE CARRERA

# Desarrollo de una aplicación Web para control de versiones de software

Autor: David Otero Gutiérrez
Tutor: Dr. Jesús Arias Fisteus

2 de marzo de 2011

TÍTULO: *Desarrollo de una aplicación Web para control de versiones de software.*

AUTOR: *DAVID OTERO GUTIÉRREZ*

TUTOR: *Dr. JESÚS ARIAS FISTEUS*

La defensa del presente Proyecto Fin de Carrera se realizó el día 9 de Marzo de 2011; siendo calificada por el siguiente tribunal:

PRESIDENTE: *Dr. Jesús Cid Sueiro*

SECRETARIO *Dr. Antonio de la Oliva Delgado*

VOCAL *Dr. Abelardo Pardo Sánchez*

Habiendo obtenido la siguiente calificación:

CALIFICACIÓN:

**Presidente** **Secretario** **Vocal**

# Agradecimientos

En primer lugar deseo expresar mi agradecimiento al tutor de este proyecto, Dr. Jesús Arias Fisteus, por todo el tiempo y apoyo recibido.

Asimismo, agradecer a mis compañeros de la Universidad Carlos III su apoyo, ya no sólo en este proyecto, sino por todo el tiempo que hemos convivido en la carrera. Han sido muchos años de duro trabajo que finalmente se ven recompensado, en lo personal por mantener la amistad con muchos de mis compañeros; y en lo profesional, por haber finalizado los estudios.

También agradecer a mi familia, que son los que siempre han confiado en mí y han estado apoyándome desde el primer día que empecé en la Universidad hasta el día de hoy.

Y para concluir, nombrar a las personas que deben aparecen por lo que han significado para mí durante estos años, desde mis primeros compañeros y amigos que han estado desde el inicio conmigo Roberto, Carlos y Kaputo, hasta los que por diversas situaciones han ido apareciendo en mi vida Miguel, Lola, Isidro, Javi, Ezquioga y Yankoa. En especial a Fátima Montaña de la Osa Barriga por el aguante que ha mostrado durante estos años y sobre todo en el tiempo de desarrollo de este proyecto, por toda la ayuda recibida y por todas las tardes de “terapia” que han sido necesarias para poder finalizar el proyecto.

# Resumen

En este proyecto se ha desarrollado una aplicación web que trabaja con un sistema de control de versiones. Este tipo de sistemas son empleados por los desarrolladores para facilitar el trabajo en grupo y para mantener un control de los cambios que se producen en un proyecto durante el tiempo de desarrollo del mismo.

Este proyecto posibilita a los usuarios el poder interactuar con un sistema de este tipo sin la necesidad de tener que disponer del software instalado en sus equipos.

Para trabajar con la aplicación los usuarios únicamente tienen que subir ficheros comprimidos que incluyan todos los archivos con los cambios que se han producido en el proyecto y seleccionar los que quieran almacenar en cada instante.

En cualquier momento podrán consultar el historial con los cambios que se han ido realizando en cada una de las versiones, y descargar el proyecto en el estado en el que se encontraba en un momento determinado.

La aplicación intenta facilitar el trabajo en grupo y la comunicación entre los miembros del mismo. Para ello se pueden crear grupos de desarrolladores que pueden trabajar sobre un mismo proyecto. Para ayudar en la comunicación la aplicación permite intercambiar mensajes y publicar comentarios en espacios comunes a todos los desarrolladores.

La motivación y el objetivo principal de este proyecto es desarrollar una aplicación web que permita a los usuarios trabajar con un sistema de control de versiones y alojar repositorios en un servidor sin la necesidad de tener instalado el software del sistema de control de versiones en la máquina del cliente.

# Índice general

# Índice de figuras

# Capítulo 1

# Introducción

## 1.1. Motivación del proyecto

A diario se desarrollan muchas aplicaciones, lo que supone modificación, creación y eliminación de documentos continuamente. De aquí surge la necesidad de gestionar de alguna marera todos los cambios que se van realizando en un proyecto. En un mismo proyecto suelen trabajar en paralelo varios desarrolladores, por lo que tener un lugar en la red donde alojar los proyectos, y al que todos los desarrolladores tengan acceso facilita la tarea de desarrollo.

Actualmente existen aplicaciones que ya cubren estas necesidades, pero cuyo uso no es trivial ya que es necesario estar familiarizado con ellas y con los sistema de control de versiones para poder usarlas. Es necesario conocer comandos propios del sistema de control de versiones para realizar ciertas acciones. Además de los conocimientos necesarios para utilizar estas aplicaciones también existe la obligación de tener instalado un software, en la máquina del cliente, para interactuar con la aplicación. Estos dos puntos son la principal motivación para la realización de este proyecto.

En primer lugar, lo que se quiere salvar es la barrera del conocimiento, es decir, no obligar al usuario a conocer los comandos necesarios para interactuar con una aplicación de control de versiones. Lo que se plantea es ofrecer una interfaz amigable mediante la que cualquier usuario pueda mantener un control de las versiones de sus proyectos con realizar acciones sencillas como son subir un fichero a un servidor y rellenar campos de texto en formularios

con la información que la aplicación les solicite. También se quiere facilitar el alojamiento y la publicación de estos proyectos en un servidor centralizado para que un grupo pueda trabajar paralelamente sobre ellos.

En segundo lugar, lo que se quiere conseguir es que no sea necesario instalar un software adicional en el cliente para poder interactuar con la aplicación y con el sistema de control de versiones, sino que sea el servidor el que se encargue de todas las tareas. Gracias a ésto se pueden gestionar las versiones de un proyecto sin necesidad de instalar ningún tipo de software, ya que todo lo hace el servidor, por lo que cualquier persona, desde cualquier ordenador podría tener acceso a sus proyectos y realizar cambios sobre ellos únicamente con tener conexión a la red.

## 1.2. Objetivos

La finalidad de este proyecto es desarrollar una aplicación Web que interactúe con un sistema de control de versiones para facilitar la creación, gestión y distribución de proyectos.

Concretamente, al término de este proyecto la aplicación debe presentar la siguiente funcionalidad:

- Creación de nuevos repositorios en el servidor.
- Almacenamiento de nuevas versiones en los proyectos alojados.
- Mostrar los ficheros, y los cambios realizados sobre ellos, en cada una de las versiones almacenadas.
- Descargar todos los ficheros existentes en cualquiera de las versiones de un proyecto.
- Controlar el acceso a los proyectos para permitir a determinados grupos de desarrolladores trabajar sobre un mismo repositorio.

## 1.3. Plan de trabajo

Para desarrollar el proyecto se ha fijado el siguiente plan de trabajo:

- Consultar la documentación de los sistemas de control de versiones para entender como trabajar con ellos y la funcionalidad que implementan.
- Estudiar las diferentes alternativas que existen para desarrollar un proyecto con estas características.
- Plantear el diseño y la arquitectura de la aplicación.
- Elaboración de una interfaz Web.
- Desarrollo de la aplicación para que interactúe con el sistema de control de versiones.
- Realizar pruebas sobre la aplicación y solucionar los errores que aparezcan.

## 1.4. Contenido de la memoria

La memoria del proyecto está estructura en varios capítulos que tratan diferentes aspectos relativos al diseño y la implementación del proyecto.

En el siguiente capítulo se puede ver una visión del estado de la tecnología actualmente. Se describen brevemente las tecnologías con las que se podría llevar a cabo el desarrollo de este proyecto, y también se mencionan otras aplicaciones que presentan una funcionalidad similar a la de este proyecto.

Una vez presentadas las posibles tecnologías que se pueden utilizar, se fijan los requisitos que debe tener la aplicación. En este capítulo se presenta un listado con toda la funcionalidad que se va a desarrollar en la aplicación.

Fijados los requisitos, se realiza un esquema de la aplicación, en el que aparecen todos los elementos que intervienen, sin entrar en detalle de las tecnologías que se utilizan para cada nodo del sistema. En este capítulo se

habla de las diferentes interfaces y los elementos que intervienen en una aplicación cliente-servidor como es el caso de este proyecto.

Tras estos capítulos, comienzan a describirse los detalles de la aplicación. En el capítulo de diseño se presentan los diferentes módulos en los que se subdivide la aplicación y que se van a integrar en ésta para cumplir con los requisitos fijados en capítulos anteriores.

El diseño del sistema ofrece una idea de como es la aplicación y la funcionalidad que ofrece, y en el siguiente capítulo, el de la implementación, se explica como se han desarrollado los diferentes módulos. En este capítulo se habla de las tecnologías que se han seleccionado para realizar las diferentes tareas tanto en el servidor como en el cliente. Se mencionan los puntos más importantes de la aplicación.

Tras las secciones en las que se habla de la implementación de cada modulo, aparece otro capítulo que presenta un listado con las pruebas que ha pasado la aplicación. En este capítulo se presentan algunos errores que se detectaron al pasar estas pruebas, y las medidas que se tomaron para solucionarlas.

En el siguiente capítulo se explican las etapas por las que se han pasado durante el desarrollo del proyecto, desde el planteamiento inicial hasta la consecución de los objetivos. Tras este capítulo, aparece otro que incluye las conclusiones y los trabajos que se deberían llevar a cabo en un futuro para seguir mejorando el proyecto.

Finalmente aparecen unos anexos en los que se explican todos los pasos que hay que seguir para instalar la aplicación en un servidor. También aparece una guía de funcionamiento, en la que se describen las vistas de la aplicación y las tareas que se pueden realizar desde cada una de ellas.

# Capítulo 2

# Estado del arte

En este capítulo se explican tres aspectos importantes para el desarrollo del proyecto. Primero se presentan los lenguajes de programación existentes para desarrollar una aplicación web, listando los más utilizados. Luego se presentan los diferentes sistemas de control de versiones existentes, mencionando algunos de ellos. Finalmente, se habla de aplicaciones web que trabajan actualmente con sistemas de este tipo.

## 2.1. Tecnologías para el desarrollo de aplicaciones para la Web.

### 2.1.1. Presentación de la información: HTML y CSS.

Estas dos tecnologías son la base de las aplicaciones Web. Desde que se publicó HTML en el año 1991 surgió la necesidad de crear un estándar para su evolución. Por eso, en el año 1994 se fundó el W3C (*Consorcio World Wide Web*) [29]. Se trataba de un consorcio de empresas y universidades de todo el mundo que se encargaría del desarrollo y estandarización de HTML. Para llevar a cabo esta tarea de estandarización existía un grupo de trabajo de HTML. Desde el año 1997 este grupo de trabajo se separó en tres secciones: una para HTML, una para DOM y otra para CSS. Desde entonces se trabaja simultáneamente en la evolución tanto de HTML como CSS.

### HTML (*Lenguaje de Marcado de Hipertexto*)

El lenguaje HTML surgió en el año 1989, cuando Tim Berners-lee propuso un nuevo sistema de hipertexto para el intercambio de información entre investigadores del CERN (*Organización Europea de Investigación Nuclear*). En el año 1991 se publicó HTML para que todos los usuarios de Internet tuviesen acceso a él. Se publicó bajo el nombre de "HTML Tags". Debido a la acogida que tuvo, y a la necesidad de tener que evolucionarlo, el organismo IETF(*Internet Engineering Task Force*) propuso la realización de un estándar para HTML. Tras dos propuestas sin éxito consiguieron, en el año 1995, que el grupo de trabajo de HTML publicase el estándar HTML2.0, convirtiéndose así en el primer estándar oficial de este lenguaje.

A partir del año 1997 el W3C se convirtió en el organismo de estandarización, y se encargó de continuar con el desarrollo del estándar HTML. La primera recomendación de este organismo fue denominada HTML3.2 y publicada en Enero de 1997. Uno de los cambios más importantes que incorporaba era la estandarización de las tablas. El borrador de HTML4 aparece en Julio de 1997, pero no se oficializó hasta finales del mismo año. Este estándar incluía mejoras para macros, hojas de estilos y permitía la utilización de *scripts* en páginas Web. En Diciembre de 1999 aparece una nueva especificación HTML, la 4.01, que define HTML4.01 [38] como una versión del HTML 4. Esta nueva especificación incluye las características de versiones anteriores, resuelve problemas antiguos y añade nuevas características multimedia y de lenguajes de *scripts*. En HTML4.01 se le da mayor importancia a la accesibilidad y a la internacionalización de documentos.

Actualmente se está trabajando en la especificación de HTML5, que es una versión que combina HTML4, XHTML 1 y DOM2. La última revisión del borrador de HTML5 [40] es la del 13 de Enero de 2011. Este estándar está formado por diferentes módulos, algunos de ellos ya finalizados, por lo que hay navegadores que pueden empezar a acoplar las nuevas funcionalidades. La fecha en la que se prevé que esta especificación esté terminada es el año 2012, hasta esa fecha los navegadores trabajarán con HTML4.01. Los navegadores además de cumplir con el estándar HTML, pueden incluir sus propias etiquetas que sólo ellos pueden interpretar. Por eso a la hora de desarrollar una aplicación Web pueden aparecer problemas de compatibilidad, provocando que una aplicación funcione en un navegador, pero en otro no lo haga correctamente.

### CSS (*Cascading Style Sheets*)

La hojas de estilo se han utilizado desde el año 1970. Se utilizaba con el lenguaje SGML, que ya veía la necesidad de definir un mecanismo para establecer estilos en documentos. Pero no fue hasta la aparición de HTML cuando se le dió realmente importancia a las hojas de estilo. Las diferencias entre los navegadores generaban dificultades para presentar los contenidos. Esto llevó a la W3C a proponer la creación de un estándar para las hojas de estilo específico para el lenguaje HTML. Se recibieron nueve propuestas, y de entre todas ellas se escogieron las dos siguientes: CHSS (*Cascading HTML Style Sheets*) y SSP (*Stream-based Style Sheet Proposal*).

Entre 1994 y 1995, Hakon Wium Lie y BertBos, creadores de cada una de las propuestas, se unieron para combinar lo mejor de cada una de ellas y definieron las bases de CSS (*Cascading Style Sheets*) [39]. El W3C apostó por la estandarización de CSS y lo añadió al grupo de trabajo de HTML. La primera recomendación oficial publicada fue denominada "CSS de Nivel 1", en el año 1996. Se trataba de un mecanismo encargado de separar el contenido de la presentación. El problema de entonces estaba en que los navegadores no daban soporte completo a CSS1.

En el año 1997, el W3C reestructuró el grupo de trabajo de HTML y ésto dió lugar al grupo de trabajo de CSS. La primera recomendación publicada por este grupo fue "CSS Nivel 2", publicada en 1998.

Desde la aparición de CSS los navegadores han tenido que trabajar para dar soporte a las hojas de estilo. Hasta el año 2000, con la aparición de "Internet Explorer 5", no se había conseguido que un navegador diese soporte completo a CSS1. Actualmente la versión utilizada por los navegadores es CSS2.1, que es una versión aún en desarrollo. Paralelamente a CSS2.1 se está trabajando en la siguiente recomendación, "CSS Nivel 3", de la que únicamente se han presentando borradores.

CSS se emplea para dar formato a los contenidos que se muestran por pantalla. Para ello CSS utiliza un mecanismo basado en reglas. Cada regla está compuesta por:

- Un selector: se utiliza para identificar a los elementos sobre los que se aplicará el estilo que indica la regla.

- Conjunto de normas: cada norma está compuesta por dos elementos, propiedad y valor. El primero, como su nombre indica, es la propiedad que se va a añadir a los elementos sobre los que aplica la regla. Y el segundo indica el valor que tendrá esa propiedad.

### 2.1.2. Lenguajes de programación.

A la hora de desarrollar una aplicación web hay que conocer los distintos lenguajes de programación que existen para tal fin. Los lenguajes de programación web se clasifican en cuatro grandes grupos. La primera clasificación se hace entre dos de esos grupos, "lenguajes estáticos" y "lenguajes dinámicos". En los orígenes de Internet los lenguajes utilizados eran estáticos, pero debido a las exigencias de las plataformas y de las aplicaciones han surgido nuevos lenguajes, que combinados con los estáticos, han ido dando solución a las necesidades que se iban presentando. La principal diferencia entre estos lenguajes está relacionada con el tiempo de ejecución y de compilación de los mismos. En los lenguajes estáticos se tiene que precompilar el código antes de ejecutarlo. Por el contrario, los dinámicos son lenguajes de más alto nivel, que son interpretados en tiempo de ejecución y no necesitan ser compilados previamente. La segunda clasificación diferencia entre otros dos grupos, "lenguajes del lado del cliente" y "lenguajes del lado del servidor". Esta clasificación hace referencia al lugar físico donde se ejecuta la lógica de la aplicación. Puede ser que el servidor sea quien realice todas las funciones que aparecen en el código, ésto ocurre en los lenguajes del lado del servidor; o bien el navegador del usuario es el que tiene que ejecutar las instrucciones, dando lugar a los lenguajes del lado del cliente.

Actualmente para desarrollar una aplicación web, con una cierta funcionalidad, es necesario combinar el lenguaje HTML con algún otro lenguaje de programación que permita presentar el contenido de forma dinámica. Para ello hay que conocer los distintos tipos de lenguajes existentes. A continuación se presentan los diferentes lenguajes atendiendo al lugar físico donde se ejecuta el código, en el servidor o en el cliente.

Primeramente están los lenguajes ejecutados en el lado del servidor. Éstos reciben las peticiones del navegador del cliente, las procesan y ejecutan todas las funciones requeridas por la lógica de la aplicación. Tras procesar la petición se genera la repuesta, y ésta es insertada en el código HTML que será enviado posteriormente al cliente. Este modo de funcionamiento hace a

estos códigos más seguros ya que el código de la aplicación se queda en el servidor y no viaja hasta el cliente.

Como principales tecnologías de lado del servidor están lenguajes como *Phyton*, PHP, ASP, ASP.NET, Java(*Servlets* y JSP), *Ruby*, PERL, CGI y SSI:

- ASP (*Active Server Pages*) es una tecnología del lado del servidor que ha sido desarrollada por Microsoft, basada en el uso de *scripts* para la realización de toda la funcionalidad requerida. Es un lenguaje que no necesita ser compilado para ejecutarse. El lenguaje más utilizado para crear aplicaciones en ASP es *VBScript*, aunque también existen otros lenguajes, como *Perl* y *Jscript* (*Javascript de Microsoft*), para desarrollarlas. El código ASP puede ser embebido en el código HTML. Como se trata de una tecnología de Microsoft, para que funcione la aplicación es necesario tener instalados productos Microsoft como son el "*Internet Information Server*"(IIS) y un servidor Windows.

- ASP.NET [2] es una tecnología del lado del servidor que se utiliza habitualmente en el desarrollo de aplicaciones web para la plataforma Windows. Se trata de un *framework* comercializado por Microsoft, y usado por los programadores, entre otras funciones, para desarrollar sitios web dinámicos. ASP .NET, es el sucesor de la tecnología ASP. Fue lanzada al mercado mediante una estrategia de mercado denominada ".NET". Para el desarrollo de ASP.NET se pueden utilizar lenguajes como C#, VB.NET o J#. Para el funcionamiento de las aplicaciones Web se necesita tener instalado el "*Internet Information Server*" junto con "*Microsoft Framework Net*". Este es el *framework* que contiene las clases e interfaces que se utilizan para simplificar y optimizar el desarrollo de las aplicaciones .NET.

- Java(*Servlets* y JSP). JSP (*Java Server Pages*) [2] es una tecnología Java utilizada para la creación de aplicaciones Web dinámicas. Esta tecnología fue desarrollada por SUN Microsystems para ser ejecutado en el lado del servidor. El código JSP puede ser embebido en el código HTML, o bien, se pueden utilizar los *servlets* de Java para ejecutar la lógica de la aplicación. JSP tiene todas las ventajas que ofrece Java, como son la portabilidad de la plataforma y también que se trata de un lenguaje que trabaja tanto la parte de la lógica del negocio como la parte del acceso a datos del sistema.

- Python [19] es un lenguaje de programación interpretado, su código no necesita ser compilado, del lado del servidor creado en el año 1990 por Guido van Rossum. Es un código comparado habitualmente con Perl. Python puede trabajar en diversas plataformas como Windows, Linux/Unix, Mac OS X, y ha sido portado a las máquinas virtuales de Java y .NET. Es un lenguaje de código abierto que permite la creación de todo tipo de programas, incluyendo los sitios web.

- Ruby [13] es un lenguaje interpretado de muy alto nivel y orientado a objetos desarrollado en el año 1993 por el programador japonés Yukihiro "Matz" Matsumoto. Su sintaxis está inspirada en Perl, Smalltalk, Eiffel, Ada y Lisp. Es distribuido bajo licencia de software libre (GPL). Ruby puede ser ejecutado en cualquier plataforma Unix, Linux, Mac OS X y Windows. Se trata de un lenguaje dinámico para una programación orientada a objetos rápida y sencilla. Para desarrollar aplicaciones Web con esta tecnología se utiliza el *framework* "Ruby on Rails" [35], que es un *framework* también de código abierto.

- PERL (*Practical Extraction and Report Language*) [25] es un potente lenguaje basado en C, AWK, sed y Lisp, entre otros. Se distribuye bajo licencia de software libre, GPL. En sus inicios fue desarrollado para manipular textos, dada su facilidad para trabajar con ellos y por la potencia de sus expresiones regulares. En la actualidad se puede utilizar para muchas más tareas, destacando el desarrollo Web, la administración de sistemas y la programación de redes. La primera versión de PERL fue denominada, PERL5, y apareció en el año 1991. La última versión estable es PERL 5.12.3. Paralelamente, desde 2002 se está trabajando en el desarrollo de una nueva versión que se conoce como PERL6. La principal ventaja que presenta este lenguaje es que, al tratarse de un lenguaje que lleva muchos años en funcionamiento, existe una gran cantidad de bibliotecas y módulos que pueden ser usados por los desarrolladores en sus aplicaciones. Esta tecnología funciona en las principales plataformas como Unix, Mac OS X y Windows.

- CGI (*Common Gateway Interface*) [1] es el sistema más antiguo que existe para presentar contenidos dinámicos en las aplicaciones Web. En las aplicaciones CGI, el servidor recibe las peticiones del cliente y las envía a una aplicación externa. Esta genera la respuesta que se envía al cliente. Los CGI se escriben habitualmente en el lenguaje Perl, aunque también se pueden emplear otros lenguajes como C, C++ o Visual Basic. Actualmente se encuentra un poco desfasado por diversas

razones entre las que destaca la dificultad con la que se desarrollan los programas y la pesada carga que supone para el servidor que los ejecuta.

- SSI (*Server Side Includes*) [18] es un antiguo lenguaje interpretado del lado del servidor que permite una limitada funcionalidad en la generación de código HTML. El uso principal de este lenguaje es el de incluir ficheros en el contenido HTML.

- PHP (*PHP Hypertext Preprocessor/Personal Home Page*) [2] es un lenguaje interpretado del lado del servidor que permite generar dinámicamente páginas web. Debido a la importancia de este lenguaje en el desarrollo del proyecto se trata con mayor detalle en la sección 2.1.3.

Por otro lado están los denominados lenguajes del lado del cliente. Estos son lenguajes interpretados y ejecutados por los navegadores, no necesitan ser pretratados para poder ser ejecutados por éstos. Entre este tipo de tecnologías destacan Javascript, los *applets* de Java, VBScript, FLASH y CSS:

- Javascript es un lenguaje de programación utilizado para crear pequeños programas encargados de realizar acciones dentro del ámbito de una página web. Se usa mayoritariamente en programación Web en el lado del cliente, porque es el navegador el que soporta la carga de procesamiento. Se emplea para generar efectos en las aplicaciones Web y para interactuar con el usuario, mediante el control de eventos. En la actualidad existen unas amplias bibliotecas de código, como por ejemplo "jQuery" [11], que simplifican el uso de este lenguaje, pudiéndose generar muchos efectos con pocas líneas de código. Existen gran cantidad de *plugins* desarrollados con jQuery que fácilmente pueden ser incluidos en las aplicaciones Web.

- Los *applets* de Java [27] están escritos en Java y son ejecutados en el cliente en el ámbito de la página web. Se ejecutan en un navegador web utilizando la máquina virtual de JAVA (JVM). Los *applets* llegan al cliente precompilados, es por ello que la manera de trabajar de éstos varía un poco con respecto a los lenguajes de *script* como Javascript. Son más difíciles de programar que los *scripts* en Javascript, y requieren de unos conocimientos básicos o medios del lenguaje Java. La principal ventaja de utilizar *applets* es que son mucho menos dependientes del navegador que los *scripts* en Javascript, además de ser independientes del sistema operativo de la máquina donde se ejecutan.

- VBScript [28] (*Visual Basic Script*) es un lenguaje de programación de *scripts* del lado del cliente, compatible con Internet Explorer y otros sistemas Microsoft. Se trata de un lenguaje interpretado por el *Windows Scripting Host* de Microsoft. Sólo puede utilizarse con navegadores de Microsoft, funcionando de forma similar a JavaScript.

- Flash no es un lenguaje de programación, sino un programa de creación de efectos y diseños especiales en páginas web. Lo que sí es un lenguaje de programación es "ActionScript", que es el lenguaje de programación de Flash. Gracias a esta tecnología se pueden crear aplicaciones Web del lado del cliente. Para visualizar las "películas" Flash el navegador debe tener instalado un *plugin* que le permita visualizarlas.

- CSS (*Cascading Style Sheets*) no se considera tampoco un lenguaje de programación, sino un lenguaje de hojas de estilo. Esta tecnología se presenta en mayor detalle en la sección 2.1.1

### 2.1.3. PHP(*PHP Hypertext Pre-processor*)

Para desarrollar esta aplicación se ha empleado el lenguaje PHP [24]. Este lenguaje fue desarrollado por Rasmus LedFord en 1994 y fue denominado en sus orígenes "Personal Home Page Tools". En la actualidad se le conoce como "PHP Hypertext Pre-processor", y es el "PHP Group" el encargado de continuar con su evolución. Se trata de un lenguaje interpretado del lado del servidor. Al ser interpretado no se compila, sino que existe un intérprete encargado de leer y ejecutar las instrucciones contenidas en el código. El uso más extendido de PHP es el de la creación de páginas web dinámicas. Este lenguaje está publicado bajo licencia de software libre, pudiendo ser desplegado en la mayoría de los servidores web, y siendo compatible con casi todos los sistemas operativos. Es en UNIX donde se obtiene la mayor eficiencia, ya que inicialmente fue desarrollado para este tipo de entornos.

Desde su aparición en 1994 han surgido cuatro ramas de desarrollo. El antecesor de PHP fue un programa llamado PHP/FI. Este programa era un conjunto de *scripts* escritos en C mediante los que se controlaba el acceso de los usuarios a una aplicación web. Rasmus Lerdorf fue desarrollando este lenguaje añadiéndole nuevas funcionalidades, hasta que finalmente, en Noviembre de 1997, se liberó la segunda versión conocida como PHP/FI 2.0. Esto permitió que cualquier desarrollador pudiera contribuir en el desarro-

llo del lenguaje. Fue así como se empezó a evolucionar hacia las siguientes versiones de PHP.

La primera de todas las versiones de PHP surge en el año 1997. Fue creada por dos desarrolladores israelíes, Zeev Suraski y Andi Gutmans, y fue denominada PHP3. Estos dos desarrolladores oficializaron esta versión del lenguaje, convirtiéndose en sucesor del antiguo PHP/FI. Este hecho hizo que se abandonase el desarrollo de PHP/FI. Fue a partir de esta versión cuando PHP pasó a ser conocido como "PHP Hypertext Pre-processor'. Tras un tiempo de pruebas, esta rama fue liberada oficialmente en Junio de 1998. Esta versión ya incluía gran cantidad de bibliotecas de código, soporte para base de datos y programación orientada a objetos.

La segunda rama de desarrollo es PHP4. Esta versión surge por la necesidad de conseguir mejoras en la ejecución de las aplicaciones y en la modularidad del código base. Para todo ello se incorpora el uso del "motor Zend" [22]. Se trata de un intérprete de código PHP, desarrollado por "Zend Tecnologies", que acelera la interpretación de los *scripts*. Este motor cambia la forma de ejecutar los *scripts*, ahora primero se compila el código y luego se ejecuta, mientras que en la versión anterior, se interpretaba y se ejecutaba de forma simultánea. Este cambio supuso que PHP fuese más rápido y más independiente del servidor en el que estuviese trabajando. Además de las mejoras de rendimiento, se incluían mejoras en el control de acceso de los usuarios, en las sesiones HTTP y mejoras de compatibilidad con la mayoría de los servidores web. Esta versión fue liberada en Mayo de 2004, y continuó en desarrollo hasta Julio de 2007.

La tercera rama fue publicada en Julio de 2004, y es conocida como PHP5. Actualmente continúa en desarrollo, aunque las nuevas actualizaciones suelen ser únicamente mejoras de seguridad. Esta versión utiliza el motor "Zend Engine 2.0", que contiene nuevas opciones y un nuevo modelo de objetos. Con esta nueva publicación, PHP ofrece importantes mejoras en la Programación Orientada a Objetos, en la conexión con bases de datos, en el soporte para XML y también incluye el manejo de excepciones. La última versión estable es la versión PHP 5.3.5 [16], publicada el día 6 de Enero de 2011. Esta versión presenta mejoras en temas de seguridad respecto a versiones anteriores.

Por último, hay que mencionar la última rama que se conoce como PHP6. Esta rama se encuentra actualmente en desarrollo y aún no ha sido publicada. Cuando se lance esta rama se abandonarán por completo las ramas anteriores a PHP5.

Para la creación de aplicaciones web los desarrolladores se apoyan en lo que se conoce como *framework* [20]. Se trata de un software que facilita el desarrollo de código de manera limpia y ordenada. Para trabajar con PHP existen diversos *frameworks* [3] que dan soporte para programación orientada a objetos y para la implementación del MVC (modelo, vista y controlador). Entre ellos destacan los siguientes: CodeIgniter [8], Akelos. [5], CakePHP [7], Kohana [36], PHP on Trax [17], Symfony [30], Prado [33] y ZendFramework [37].

Esta aplicación web ha sido desarrollada utilizando el *framework* conocido como "ZendFramework" [37]. Se trata de un *framework* de código abierto para el desarrollo de aplicaciones en PHP5. Surge en el año 2005 creado por la empresa "Zend Technologies", interviniendo en su desarrollo dos de los creadores de PHP. La última versión estable es la 1.11.3, publicada en Enero de 2011. Este *framework* tiene como principios básicos la programación orientada a objetos y la implementación del MVC para el desarrollo de aplicaciones. Gracias a que el código se separa en cada uno de los tres niveles del modelo, vista y controlador, se facilita el desarrollo de las aplicaciones, ganando tiempo y claridad en la programación.

*ZendFramework* dispone de una extensa biblioteca de módulos y funciones que se pueden combinar para obtener una gran funcionalidad. Entre todos los módulos los hay que facilitan tareas como la autenticación de usuarios, el control de sesiones, la creación de aplicaciones multilenguaje y la gestión e interacción con las bases de datos. *ZendFramework* soporta gran cantidad de sistemas de bases de datos entre los que se incluyen MySQL, Oracle, SQLite, Microsoft SQL Server e IBM DB2.

La empresa creadora de *ZendFramework* también ha desarrollado otros dos programas, *ZendServer* y *ZendStudio*, que se pueden utilizar conjuntamente con las aplicaciones desarrolladas con este *framework* para optimizar su uso. *ZendServer* es un servidor que consigue mejorar el rendimiento de aplicaciones desarrolladas con *ZendFramework*; y *ZendStudio* es un entorno de desarrollo que incluye las bibliotecas de PHP. Para desarrollar aplicaciones web no es necesario disponer de ninguno de estos dos programas, únicamente hay que tener instalado el *framework*.

## 2.2. Sistemas de control de versiones.

Un sistema de control de versiones (VCS) es un *software* que se utiliza para almacenar las diferentes versiones por las que pasa un proyecto durante su desarrollo. En cada una de éstas habrá ficheros nuevos, modificados o eliminados. Los sistemas de control de versiones se encargan de gestionar los diferentes estados por los que pasa una aplicación durante todo el período de desarrollo, guardando un historial con todos los cambios realizados entre las versiones. Esto facilita el desarrollo de proyectos ya que se pueden ir guardando versiones a medida que se van cumpliendo objetivos de la aplicación, y posteriormente poder recuperar cualquiera de ellas si fuera necesario. También, gracias a ésto se consigue que varias personas puedan trabajar paralelamente sobre un mismo proyecto, y puedan publicar cada uno los cambios que han realizado para que el resto de desarrolladores tengan acceso a la última versión disponible.

A continuación se citan las principales funciones que ofrece un sistema de control de versiones:

- Almacenar en un historial los cambios realizados sobre un proyecto.
- Gestionar todas las versiones de un proyecto.
- Crear y eliminar versiones.
- Mostrar las diferencias entre versiones.
- Recuperar versiones antiguas deshaciendo todos los cambios realizados desde la versión anterior hasta la actual.
- Crear ramas de trabajo sobre un proyecto en desarrollo para trabajar en paralelo.

Trabajar con este tipo de *software* también facilita la distribución de contenidos, ya que se pueden compartir los proyectos entre varios desarrolladores, y éstos podrían trabajar paralelamente sobre un mismo proyecto, manteniendo una copia con todos los cambio en un servidor o en una máquina local.

Los sistemas de control de versiones se clasifican en "sistemas centralizados" y "sistemas distribuidos". La principal diferencia entre ellos es la

forma en la que se comparten los proyectos. En los centralizados, el historial de cambios del proyecto se mantiene en un único repositorio de código. Por el contrario, en los distribuidos se puede hacer una copia local de todo el proyecto, incluyendo el historial de cambios. Con ésto se consigue que se puedan intercambiar distintas versiones de un mismo proyecto entre varios repositorios, sin necesidad de que haya un servidor central.

### 2.2.1. Sistemas centralizados.

En los sistemas de control de versiones centralizados (CVCS), el proyecto y todos sus cambios se almacenan en un servidor central que aloja el repositorio del proyecto, por lo que toda la información con el historial de cambios estará únicamente en este servidor. Este tipo de sistemas se utiliza cuando múltiples personas colaboran sobre un mismo proyecto. Cuando un desarrollador quiere hacer alguna modificación debe conectarse a ese servidor, descargarse una copia a su equipo, y sobre ésta realizar los cambios pertinentes. Tras realizar todas las modificaciones sobre la copia local se envía de nuevo al repositorio original, situado en el servidor central. El CVCS guardará la nueva versión en el historial de cambios, y así el resto de desarrolladores tendrán acceso a la última versión.

Todos los desarrolladores trabajan sobre una rama o ramas pero en un mismo servidor central. El sincronismo de las fuentes se hace siempre desde el servidor central, y es en éste donde se encuentra la versión más reciente del proyecto. Para poder trabajar con este tipo de VCS es necesario tener conexión de red, ya que cuando se realicen cambios hay que actualizar el servidor central para que los demás puedan trabajar sobre el proyecto. Este es el mecanismo que emplean los sistemas de control de versiones centralizados como SCV y Subversion [32]. El principal inconveniente que pueden presentar este tipo de sistemas es que se depende de un servidor. Cualquier caída de éste supondría un retraso en el proyecto. Otro inconveniente es que existe un único repositorio del proyecto. La figura 2.1 ilustra el esquema de trabajo de un sistema de control de versiones centralizado, en el que cada usuario realiza cambios y crea nuevas versiones en el repositorio central.

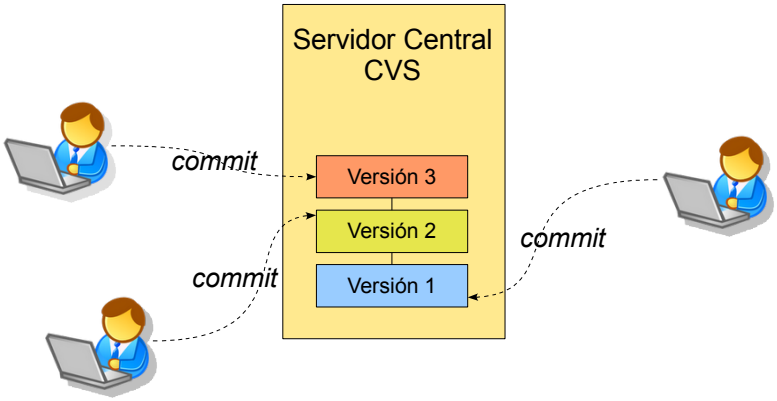


Figura 2.1: Sistema de control de versiones centralizado.

### 2.2.2. Sistemas distribuidos.

Por otro lado están los sistemas de control de versiones distribuidos (DVCS). Normalmente, estos VCS también parten de un repositorio principal al que todos los desarrolladores tienen acceso. En cambio, en este tipo de sistemas se permite que cada desarrollador pueda trabajar en una rama independiente y de manera local.

Para trabajar con ellos lo primero que se hace es clonar en la máquina local el repositorio del proyecto. Esto da lugar a un repositorio completo, que incluye todos los ficheros del proyecto, el historial de cambios, etc. Gracias a ésto, cada desarrollador puede trabajar paralela e independientemente, e ir guardando sus propias versiones en un repositorio local. Cuando lo crea conveniente podrá sincronizar su repositorio local con el repositorio remoto alojado en el servidor. Para ello se enviarán al servidor tanto los ficheros, como el historial de los cambios realizados en el repositorio local. Este tipo de VCS también permite separar las tareas de desarrollo de un proyecto en diferentes ramas de trabajo. Cada rama estará orientada a desarrollar diferentes aspectos de la aplicación. Finalmente, el VCS se encargará de combinar las diferentes ramas, de manera que quede una única copia con todas los cambios realizados en las ramas de trabajo.

Con este tipo de VCS ya no es necesario tener acceso a la red para poder trabajar sobre un proyecto. Ahora se puede trabajar sobre un repositorio

local, ya que se tiene acceso tanto a los ficheros como al historial del proyecto. Esto permite que se puedan crear nuevas versiones en la copia local. Los desarrolladores pueden sincronizar la copia local con la remota en cualquier instante, enviando sus cambios al repositorio remoto, o descargando los cambios que se hayan añadido en el servidor.

A la hora de distribuir un proyecto los DVCS permiten que los desarrolladores compartan sus proyectos sin tener que pasar antes por un servidor central, pudiendo compartir los repositorios locales directamente entre ellos. Entre los DVCS destacan sistemas como GIT, Mercurial, Darcs, Monotone y Bazaar. En la figura 2.2 se observa el intercambio de versiones entre los desarrolladores y entre el repositorio central, los desarrolladores realizan cambios en sus repositorios locales y luego envían estos cambios a otros desarrolladores; finalmente, un desarrollador actualiza el repositorio remoto alojado en el servidor.

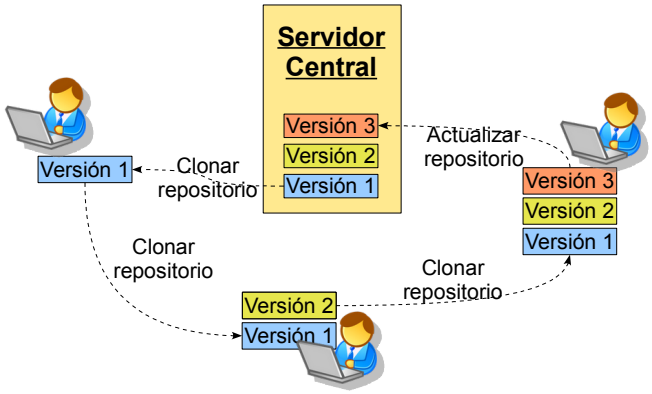


Figura 2.2: Sistema de control de versiones distribuido.

## 2.3. El sistema de control de versiones Git.

El sistema de control de versiones que se utiliza en el desarrollo de este proyecto es Git [23]. En las siguientes secciones se detallan los aspectos más relevantes de su historia, de su modo de trabajo y de la estructura de los proyectos creados a partir de él.

### 2.3.1. Historia

Git es sistema de control de versiones desarrollado por Linux Torvalds, creador de Linux, en en el año 2005. Entre los objetivos que tenía estaban el diseño sencillo, la velocidad, la posibilidad de trabajar con muchas ramas paralelamente, que fuese distribuido y que pudiese trabajar con proyectos grandes, como era el proyecto del núcleo de Linux, de manera eficiente. Este sistema de control de versiones puede trabajar en plataformas Windows, Mac OSX y Unix. La última versión estable es la v1.7.4.1, publicada en Febrero de 2011.

Gracias a la forma de trabajar de Git, la mayoría de las operaciones se realizan de manera local, sin necesidad de intervención de otros equipos de la red. Esto se puede hacer gracias a que se mantiene un copia del repositorio, ficheros e historial de cambios, en la máquina local. Esta forma de trabajar presenta una ventaja frente a otros sistemas de control de versiones, ya que no es necesario tener conexión a la red para poder seguir trabajando y guardando los cambios. Se trabaja sobre la copia local y cuando se precise se sincroniza con la copia remota.

### 2.3.2. Estructura de los repositorios.

Los repositorios creados con este VCS tienen una estructura definida por el propio sistema en la que se distinguen tres secciones de trabajo:

1. El *directorio de Git*, que es donde se guardan los objetos que mantienen el historial con los cambios que se han ido produciendo en el proyecto.
2. El *directorio de trabajo*, que contiene los ficheros de la versión actual del proyecto sobre los que se realizan los cambios.
3. El *área de preparación* o índice, que se trata de un fichero que incluye la información de los cambios que se van a enviar en la próxima confirmación.

### 2.3.3. Flujo de trabajo.

El primer paso para comenzar a trabajar con Git es crear un repositorio. Esto se puede hacer a partir de un directorio local o de un repositorio ya existente. Si se crea partiendo de un directorio local, Git lo que hace es añadir "el directorio de Git", con toda la información del versionado, al directorio de trabajo local. En la segunda opción, crearlo a partir de un repositorio existente, lo que hace Git es clonar el repositorio original, ya sea remoto o local, generando un nuevo repositorio local. Este repositorio local contiene toda la información del historial de cambios y los ficheros del directorio de trabajo. Los repositorios clonados son independientes del repositorio original, los cambios que se realicen sobre éstos no afectarán al original. Una vez creado el repositorio ya se puede empezar a trabajar con Git, creando y almacenando nuevas versiones del proyecto.

Una versión se puede definir como una "instantánea" del proyecto en un determinado momento. A la acción de crear y almacenar una nueva versión se le conoce con el nombre de *commit*. Para realizar un *commit* hay que seguir los siguientes pasos:

1. Consultar el estado del repositorio. Con ésto se obtiene un listado con todos los ficheros que han sido cambiados. Los archivos pueden encontrarse en estado modificado, renombrado, eliminado o añadido.

2. Seleccionar los cambios que se almacenarán en la nueva versión. Para realizar esta acción existen comandos que añaden o eliminan ficheros al área de preparación, que es donde se encuentran todos los cambios que se van a confirmar en el *commit*.

3. Escribir un mensaje corto y descriptivo de los cambios que se han realizado. A la hora de elaborar este mensaje es conveniente seguir un formato establecido, que consta de una línea con el título de la versión, seguido de una línea en blanco y a continuación unas líneas que describan los cambios que se han incluido en la nueva versión. Este mensaje puede servir al resto de desarrolladores para identificar las versiones, aunque el identificador real de un *commit* es un *hash* SHA-1. Este *hash* se genera mediante comprobaciones en los datos de cada *commit*. Es una cadena de 40 caracteres de longitud que además de emplearse para identificar el *commit* sirve para verificar que la información es correcta y no hay datos corruptos tras hacer el *commit*.

Tras todos estos pasos se almacena una nueva copia permanente en el historial del proyecto con los cambios incluidos en el área de preparación. Cuando se confirman los nuevos cambios, Git almacena una nueva copia de todos los ficheros modificados, y de los que no han sido modificados únicamente guarda un enlace al fichero anterior que ya estaba almacenado en el repositorio del proyecto. En la figura 2.3 se puede ver el flujo de trabajo de Git a la hora de realizar un nuevo *commit*.

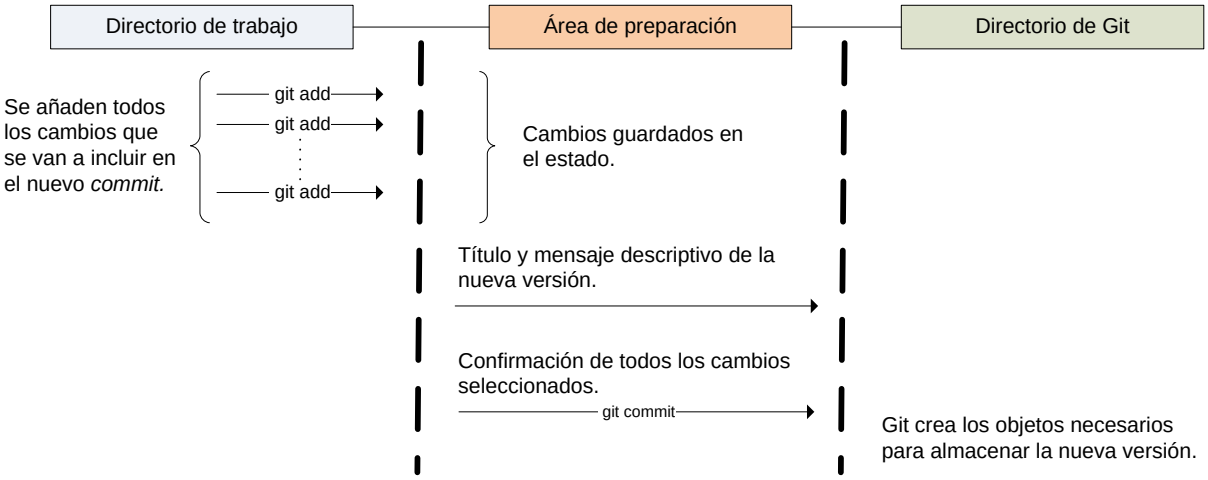


Figura 2.3: Flujo de trabajo de Git, creación de una versión.

### 2.3.4. Trabajando con ramas.

Una de las características más importantes de Git es la posibilidad de trabajar con ramas. Una rama es un punto de trabajo paralelo sobre un mismo proyecto. Gracias a la creación de una rama se puede trabajar de manera paralela e independiente sobre un mismo proyecto ya que los cambios que se realicen sobre la nueva rama no afectan al resto del proyecto. Git crea por defecto una rama maestra cuando se inicia un nuevo repositorio. Esta rama suele usarse como si de un nodo central se tratase, siendo el punto de sincronismo con el resto de ramas. Cuando se crea una nueva rama, en ésta se copia todo el historial del proyecto, por lo que desde esa rama se puede seguir trabajando de forma independiente, creando nuevas versiones. Cada vez que se realiza un *commit* en una rama, sólo ésta se ve afectada.

La idea de trabajar con ramas es la de separar las diferentes tareas que se desarrollan sobre un proyecto para poder realizarlas simultáneamente. Pero

el separar un proyecto en ramas lo que implica es que en un futuro todas estas ramas de desarrollo deberán converger en un punto final. Esta es la idea con la que trabaja Git, y para ello facilita la posibilidad de combinar ramas. Una vez hechos los cambios sobre una rama se pueden volcar éstos a la rama de origen, o a la rama que se quiera. Ésto se realiza mediante una acción que se conoce como *merge*. Esta acción lo que hace es combinar los historiales y ficheros de las dos ramas implicadas. Git detecta todos los cambios que hay en las dos ramas y es capaz de combinarlos, dejando una única versión con los cambios que se han realizado sobre ambas. Al combinarlas se cambian todos los ficheros, en la rama destino, que estén modificados en la rama origen, generando una nueva versión a partir de ambas ramas.

En la figura 2.4 se puede ver gráficamente como funciona el trabajo con ramas. En ella se ven dos ramas que se "separan" de la rama *master* en un instante determinado, y sobre las que se hacen una serie de *commit* para terminar combinándose de nuevo con la rama *master*.

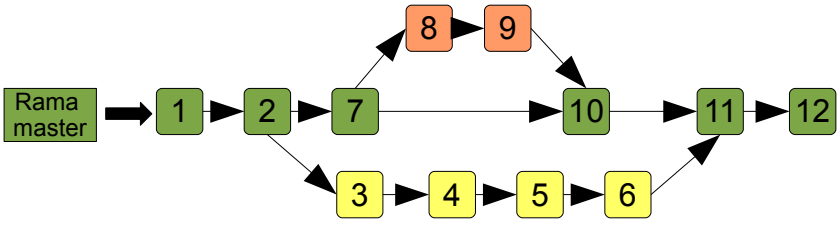


Figura 2.4: Trabajando con ramas en Git

## 2.4. Trabajos relacionados.

Actualmente ya existen aplicaciones Web que ofrecen un lugar en la red para alojar proyectos. En ellas se crean repositorios remotos que serán accesibles por desarrolladores de todo el mundo. Para acceder a los repositorios, únicamente es necesario tener un sistema de control de versiones instalado en una máquina local. Los repositorios remotos se publican bajo una *url*, que será a la que se conectarán los sistemas de control de versiones para realizar las acciones requeridas.

Desde cualquier navegador Web se puede acceder a estas aplicaciones para

realizar cualquiera de las siguientes acciones:

- Ver un listado de los proyectos alojados en el servidor.
- Consultar el historial con las datos de los diferentes *commits* que se han realizado en un repositorio.
- Ver el árbol de ficheros y directorios de una versión determinada.
- Ver las diferencias de un fichero respecto a alguna versión anterior.

Existen diversas aplicaciones de este tipo que permiten trabajar con sistemas de control de versiones [21]. Las más extendidas son las que trabajan con "Git" y con "Subversion". Entre las aplicaciones que utilizan "Subversion" destacan, por el número de proyectos que alojan y los usuarios registrados, "Google Code" y "CodePlex". En cuanto a las aplicaciones que se basan en el uso de Git destaca por encima de todas "GitHub", aunque existen otras como "Gitorious", "BerliOS", y "GNU Savannah". También existe alguna aplicación, como "SourceForge", que es capaz de trabajar con varios sistemas de control de versiones. Para interactuar con estas aplicaciones es necesario tener instalado un programa de control de versiones en una máquina local, y será desde ésta, por medio de *ssh* u otro protocolo, como se sincronicen los repositorios locales con los remotos.

Todas las aplicaciones comparten la funcionalidad indicada arriba, algunas además incluyen soporte para el registro y la gestión de usuarios. Con ésto se consigue que se puedan crear repositorios públicos o privados, para limitar el acceso de los desarrolladores a los proyectos. En función de si el repositorio creado es público o privado, la aplicación será gratuita o no.

De entre todas las aplicaciones que trabajan con Git, la más completa se llama "GitHub". Su funcionalidad incluye más tareas que el resto de aplicaciones Web de alojamiento de repositorios. *GitHub* se puede definir como una red social para desarrolladores. En 2009 esta Web contaba con 47.000 [9] repositorios públicos, y a día de hoy, cuenta con 478.000 usuarios alojando un total de 1.448.000 repositorios. Se trata de un servidor dedicado al alojamiento y distribución de proyectos. Los proyectos que aloja pueden tener tanto acceso libre como acceso privado. Si el acceso es libre, la aplicación es gratuita, por contra, si se quiere un acceso privado, es necesario pagar para recibir este servicio.

Entre las principales tareas que se pueden realizar con esta aplicación web destacan:

- Alojar repositorios, tanto públicos como privados, para que sea accesibles desde cualquier lugar de la red.
- Gestionar los cambios que se producen sobre los repositorios, permitiendo que se puedan subir cambios. A la hora de enviar cambios se puede permitir que se suban directamente o que exista una cola donde se almacenarán los cambios que deben ser confirmados por el administrador del proyecto para que tengan efecto.
- Presentar el contenido de un proyecto de forma gráfica. Mostrando las diferentes versiones, las diferentes ramas y toda la información relacionada con el proyecto. Cuenta con una herramienta gráfica más elaborada.
- Permite seguir el trabajo de otros desarrolladores, consultando los trabajos en los que están implicados, viendo cuáles han sido los últimos cambios que han hecho, enviar mensaje a otros desarrolladores, etc.
- Empaquetar y descargar el contenido de un proyecto en un único fichero.
- Se pueden incluir comentarios en el código de los proyectos y colaborar en los *wikis*, aportando nuevas ideas y comentarios sobre los proyectos.

Para empezar a trabajar con *GitHub* hay que tener instalado Git en una máquina local. Lo primero que hay que hacer es crear una clave pública para poder conectarse mediante SSH desde la máquina local al servidor de *GitHub*. Una vez creada la clave pública, se copia ésta en el perfil del usuario a través de la aplicación Web. El siguiente paso que hay que hacer es crear el repositorio remoto, ésto se puede hacer desde línea de comandos en una máquina local, o por la interfaz Web de la aplicación. Una vez creado el repositorio ya se puede subir el contenido al mismo. Para subir los cambios al servidor únicamente se necesita conocer la *url* donde está publicado el repositorio remoto. Los cambios se envían al servidor mediante comandos de Git ejecutados por línea de comandos en la máquina local.

Para trabajar con estas aplicaciones Web hay que cumplir con uno requisitos comunes a todas ellas. El primero es la necesidad de tener un sistema

de control de versiones instalado en la máquina local, desde la que se conecta con el servidor web, para poder interactuar con las aplicaciones. En este proyecto, uno de los objetivos es eliminar este requisito, ya que es el servidor el único que debe tener instalado Git. Con ésto se consigue que desde cualquier máquina, sin necesidad de instalar el *software* del sistema de control de versiones, se puede trabajar con Git. Únicamente con subir al servidor los ficheros que se quieren guardar, mediante el uso de las interfaz gráfica, se pueden mantener las versiones de un proyecto.

Otro requisito que presentan estas aplicaciones es que hay que tener conocimientos de los sistemas de control de versiones para poder trabajar con ellas, ya que su uso no resulta trivial. En cambio, este proyecto tiene como objetivo simplificar el trabajo con sistemas de este tipo, que el usuario no necesite tener amplios conocimientos en estos sistemas para poder trabajar sin problemas con la aplicación Web.

# Capítulo 3

# Requisitos

La aplicación tiene que dar soporte a toda la funcionalidad que ofrece un sistema de control de versiones y añadir cierta funcionalidad adicional necesaria en una aplicación web. Por tanto, la aplicación debe centrarse en la consecución de la siguiente funcionalidad:

- Requisito 1: el sistema debe permitir el registro de usuarios y controlar el acceso a la misma únicamente de usuarios registrados.
- Requisito 2: el sistema debe crear los repositorios donde se desarrollaran los proyectos. Cada repositorio tendrá asociada una lista de usuarios con acceso al mismo.
- Requisito 3: el sistema debe permitir trabajar con diferentes ramas sobre un mismo repositorio. Para ello se deben poder crear ramas y realizar *merge* entre ellas, es decir, poder combinar los cambios entre diferentes ramas.
- Requisito 4: el sistema debe permitir realizar nuevos *commits* en cualquiera de los repositorios alojados en el servidor, siempre que el usuario tenga acceso al proyecto.
- Requisito 5: el sistema debe mostrar el historial con todos los cambios y las versiones de un proyecto, y permitir realizar tareas de recuperación, de comparación y de eliminación de versiones.
- Requisito 6: el sistema debe permitir acceder al contenido de los ficheros y directorios en cualquiera de las versiones, mostrando las diferencias entre cada versión.

- Requisito 7: el sistema debe permitir descargar el contenido que tiene un proyecto en una determinada versión, creando para ello un fichero comprimido con todos los archivos y directorios de dicha versión.
- Requisito 8: el sistema debe facilitar el intercambio de mensajes y de comentarios entre los usuarios.
- Requisito 9: el sistema debe ser multilenguaje, debe presentar el contenido en varios idiomas permitiendo al usuario escoger entre ellos.

# Capítulo 4

# Arquitectura del sistema

## 4.1. Arquitectura software

La figura 4.1 muestra la arquitectura de la aplicación. Sin entrar en detalles de como interactúan unos elementos con otros se observan cuatro entidades con la siguiente función:

1. Los usuarios: son los que solicitan y envían la información necesaria para que la aplicación realice las tareas que deba en cada momento. Interactúan con el sistema realizando peticiones *http* desde sus navegadores al servidor.

2. El servidor Web: es el que recibe las peticiones *http* de los usuarios, las procesa e interactúa, tanto con la base de datos como con el sistema de control de versiones, para generar y enviar la respuesta a los usuarios.

3. El sistema de control de versiones Git: es el software desplegado en el servidor que se encarga de la creación y del mantenimiento de todas las versiones almacenadas en el historial de los repositorios alojados en el servidor.

4. La base de datos: es donde se almacena la información adicional que se necesita para el funcionamiento de la aplicación Web. Interviene en tareas como el control de acceso y el intercambio de mensajes entre usuarios, y también almacena información de los proyectos creados.

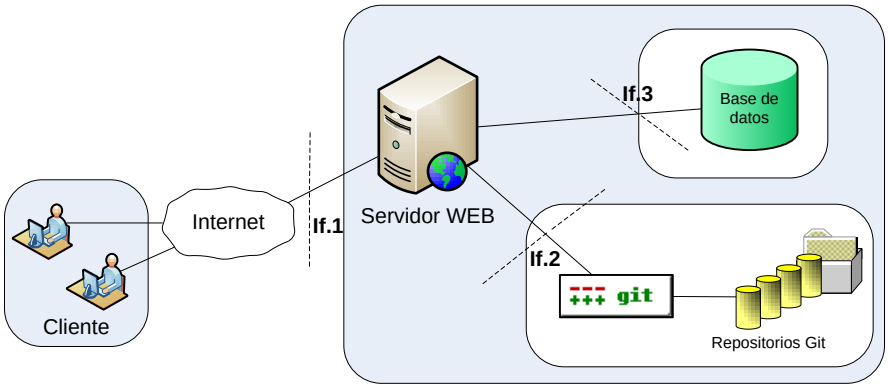


Figura 4.1: Arquitectura software de la aplicación.

## 4.2. Interfaces

Como se observa en la figura 4.1 existe un nodo central, el servidor Web, que se encarga de interconectar el resto de elementos de la arquitectura. Por tanto, las interfaces existentes son las siguientes:

- If.1: acceso Web de los clientes a la aplicación.
- If.2: conexión entre la aplicación Web y el sistema de control de versiones Git.
- If.3: conexión con la base de datos.

### 4.2.1. If.1 : Acceso Web de los clientes.

Los usuarios podrán acceder a la aplicación Web desde cualquier navegador Web mediante una IP pública. La aplicación cuenta con un mecanismos de autenticación, por lo que para acceder a la aplicación el usuario deberá estar registrado y autenticado correctamente. Una vez que un usuario se valida, accede a la vista inicial de la aplicación y puede empezar a trabajar sobre los proyectos.

### 4.2.2. If.2 : servidor Web y sistema de control de versiones.

Este es el núcleo de la aplicación. El servidor Web es el único que va a interactuar con el sistema de control de versiones. La aplicación Web es la encargada de realizar las tareas de lectura y escritura que los usuarios deseen para gestionar los repositorios. Para llevar a cabo todas estas tareas, la aplicación hace uso de los comandos y opciones propias del sistema de control de versiones.

### 4.2.3. If.3: conexión con la base de datos.

La aplicación Web se conecta a la base de datos para almacenar información adicional necesaria para el funcionamiento de la aplicación. Se almacenan datos relativos a los usuarios, a los proyectos y a los mensajes intercambiados entre los usuarios. Por tanto, la aplicación Web cuenta con una conexión a la base de datos, sobre la que se realizarán las consultas necesarias para controlar el acceso de los usuarios, para gestionar los repositorios y el intercambio de mensajes.

# Capítulo 5

# Diseño del sistema

## 5.1. Módulos.

La aplicación Web se puede dividir en varios módulos atendiendo a su funcionalidad. Una parte de la aplicación es la encargada del entorno gráfico y la presentación de los contenidos; otra se encarga de interactuar con el sistema de control de versiones, y otra de la gestión del acceso de los usuarios.

A continuación se presentan los diferentes módulos en los que se puede dividir la aplicación.

### 5.1.1. Interacción entre la aplicación Web y el sistema de control de versiones.

El eje de la aplicación es el trabajo que lleva a cabo el sistema de control de versiones Git. La aplicación Web debe interactuar con Git para el mantenimiento de los repositorios. Sobre éstos se pueden realizar tareas de lectura y de escritura.

No todas las acciones que se realizan sobre los repositorios deben ser ejecutadas directamente con llamadas a Git. Hay ciertas tareas de lectura que se pueden realizar consultando los ficheros de control que Git genera, que están guardados en "el directorio de Git". Esto es lo que hace una biblioteca de código escrita en PHP llamada "Glip" [4]. Esta biblioteca lee los archivos

que Git genera para el mantenimiento del historial de los repositorios, y obtiene de ellos información relativa a los *commits* realizados, a las ramas creadas y a los ficheros y directorios contenidos en cada versión.

Como se ve en la figura 5.1, la aplicación Web tiene dos caminos para interactuar con Git. Con uno se realizan tareas de lectura y escritura, mediante llamadas por línea de comandos directamente a Git; y con el otro se realizan acciones de lectura haciendo uso de las funciones contenidas en la biblioteca "Glip".

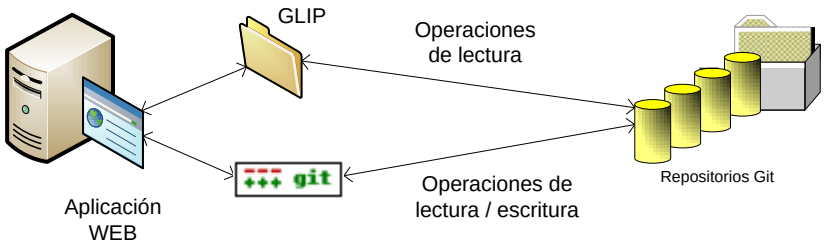


Figura 5.1: Conexiones servidor Web - Repositorios GIT.

### 5.1.2. Control del acceso y gestión de usuarios.

La primera acción que se realiza en la aplicación es la de controlar el acceso de los usuarios. Para poder trabajar con la aplicación es necesario que los usuarios estén registrados, y que se autentiquen correctamente cada vez que accedan. Para llevar a cabo la tarea de autenticación, la aplicación Web hace consultas a la base de datos, donde se almacenan los datos de las cuentas de los usuarios. Si la autenticación es correcta el usuario podrá hacer uso de toda la funcionalidad.

Este módulo, además de controlar el acceso de los usuarios, también permite a éstos editar su perfil, cambiando algunos de los datos personales que introdujeron en el momento del registro.

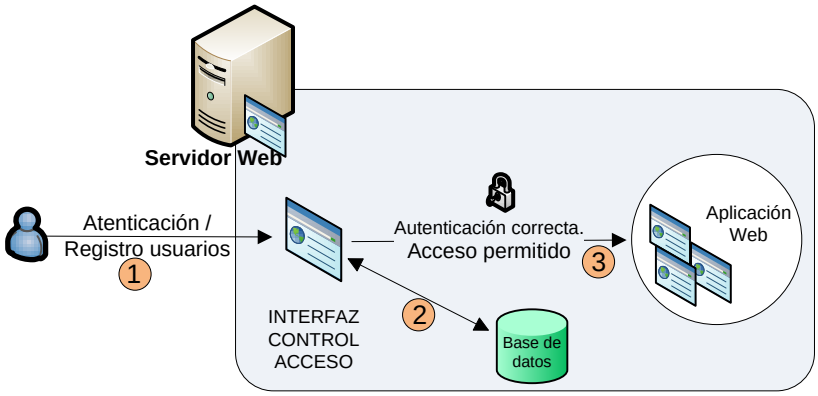


Figura 5.2: Control del acceso de usuarios.

### 5.1.3. Multilenguaje.

La aplicación web cuenta con soporte para multilenguaje. El usuario puede seleccionar en que lenguaje quiere se presenten los contenidos. Los posibles lenguajes serán inglés o castellano. Para cambiar el lenguaje el usuario tiene una opción en el menú superior de la vista principal. Cuando se cambia el lenguaje, la aplicación actualiza el perfil del usuario, almacenando el lenguaje escogido en la base de datos, para presentar todos los contenidos con ese lenguaje desde el momento de la elección.

### 5.1.4. Comunicación entre usuarios.

La aplicación ofrece a los usuarios un mecanismo mediante el que pueden comunicarse unos con otros. Los usuarios podrán enviarse y compartir comentarios de dos formas distintas :

- Mediante el envío de mensajes privados directamente de unos usuarios a otros. Un usuario puede consultar un listado con el nombre del resto de usuarios registrados y enviarles un mensaje privado, que sólo el destinatario podrá leer. La funcionalidad de este mecanismo se podría comparar con un sistema de correo electrónico.
- Existen *wikis* para que los usuarios compartan públicamente comentarios sobre los distintos proyectos. Existirá un *wiki* por cada proyecto, y también uno para cada rama dentro de un proyecto. Los usuarios

tendrán acceso a estos *wikis*, para consultarlos o editarlos, siempre que tengan acceso al proyecto.

## 5.2. Interacción con los usuarios.

La aplicación web cuenta con diversas vistas que se encargan de mostrar el contenido de los repositorios y permitir a los usuarios interactuar con ellos. Desde las vistas de la aplicación se pueden realizar todas las tareas relacionadas con la creación y gestión de los repositorios Git, el envío de mensajes entre usuarios, el uso de *wikis* para compartir información sobre proyectos con otros desarrolladores y la gestión del perfil de los usuarios.

### 5.2.1. Vista control de acceso.

En esta vista se presenta un formulario que los usuarios deberán rellenar, con su alias y contraseña, para poder autenticarse y así acceder a la aplicación. Desde esta vista, los usuarios que no estén registrados pueden crearse una cuenta rellenando el formulario de ingreso.

### 5.2.2. Vista del menú principal.

Esta es la vista que se carga al inicio de la aplicación, tras la autenticación de los usuarios. En ella aparece una tabla con los proyectos alojados en el servidor a los que el usuario tiene acceso. De cada proyecto aparece la siguiente información: nombre del autor, título del proyecto, fecha de creación y el tipo de permiso de acceso que tiene el usuario. Desde esta vista se puede acceder al contenido de cualquier repositorio.

Desde aquí también se pueden enviar solicitudes de acceso a cualquiera de los proyectos privados a los que el usuario tenga prohibido el acceso.

### 5.2.3. Vista para la creación de proyectos.

En esta vista se presenta un formulario con los datos necesarios para poder iniciar un repositorio donde guardar el nuevo proyecto. El usuario debe introducir el título del proyecto, la descripción del primer *commit* y seleccionar el fichero comprimido que contiene los ficheros que formarán el proyecto. Tras enviar el formulario se crea el repositorio en el servidor.

### 5.2.4. Vista administrar permisos.

El creador de un proyecto es el encargado de administrar los permisos de acceso a dicho proyecto. Para ello, en esta vista se presenta un listado con todos los usuarios y grupos, y el tipo de permisos que tienen, si pueden acceder o no al proyecto. Desde aquí se puede permitir o denegar el acceso a cualquier usuario o grupo. En el listado irán apareciendo las nuevas solicitudes que los usuarios vayan enviando.

### 5.2.5. Vista listado de ramas.

Esta es la vista a la que se accede al entrar dentro de un proyecto. En ella se presenta una tabla con la información de las ramas que se han creado en el repositorio. También se muestra un breve encabezado con un resumen de la información del proyecto: título, fecha de creación y autor.

Desde esta vista se pueden realizar varias acciones. Todos los usuarios podrán acceder a un *wiki* del proyecto, en donde podrán dejar comentarios durante el desarrollo del proyecto. Adicionalmente, si el que accede a esta vista es el administrador del proyecto, desde aquí puede acceder al área de administración desde donde se cambian los permisos de acceso al proyecto.

### 5.2.6. Vista historial del proyecto.

En esta vista se muestra una tabla con todos los *commits*, las diferentes versiones, que se han realizado sobre la rama del repositorio seleccionada.

Además de la información de las diferentes versiones, se muestra un breve encabezado con datos del proyecto, indicando la rama actual y el nombre del proyecto.

Desde esta vista, ademas de consultar el historial de un proyecto, se pueden realizar las siguientes acciones:

- Realizar un nuevo *commit*: para ésto se muestra un formulario con los datos necesarios para poder crear la nueva versión. En este formulario lo primero que se pide es que se seleccione un fichero comprimido, con formato "Zip" o "Rar", que contenga todos los ficheros que se quieran almacenar en el proyecto. Luego se debe introducir el nombre y la descripción de la nueva versión. La última opción que hay que seleccionar es si Git debe tener en cuenta los ficheros eliminados o no. Si se selecciona que sí, Git comparará todos los ficheros de la última versión con los que contenga el fichero que se suba, detectando tanto los ficheros nuevos y modificados, como los eliminados. Si se selecciona no detectar los eliminados, únicamente comparará los ficheros que trae el fichero comprimido con los mismos que estaban almacenados en la última versión, teniendo en cuenta únicamente los ficheros nuevos y modificados.

- Descargar una versión dada: se generará un fichero comprimido con todos los ficheros que tenía el proyecto en el estado en el que estaban en la versión que se seleccione.

- Combinar la rama actual con la rama *master*, acción conocida como "merge". Vuelca los cambios de la rama actual en la rama principal del proyecto, quedándose como la versión más reciente la misma en ambas ramas.

- Actualizar la rama actual. Acción parecida a la anterior, pero que en este caso vuelca sobre la rama actual los cambios que hay en la rama *master*, apuntando las dos ramas a la misma versión.

- También se puede acceder al *wiki* de la rama para compartir comentarios. Cada rama tiene un *wiki* independiente.

### 5.2.7. Vista confirmar cambios del nuevo *commit*.

En esta vista se presenta un árbol de directorios con todos los ficheros modificados, añadidos o eliminados. Si un usuario se encuentra en esta vista significa que está en un punto intermedio de la creación de un nuevo *commit*. En esta vista deberá seleccionar los cambios que se van a incluir en la nueva versión, y confirmar posteriormente todos estos cambios que tendrá la nueva versión, realizándose finalmente el *commit*.

### 5.2.8. Vista árbol de directorios.

Esta vista muestra los ficheros y directorios que hay en una determinada versión de un proyecto. En ella se puede seleccionar que tipo de árbol de directorios se quiere ver, existen dos estructuras de directorios distintas:

1. Se puede ver un árbol con todos los directorios y ficheros que tiene la versión seleccionada.
2. Mostrar únicamente los ficheros que han sido modificados o añadidos desde la versión anterior, ignorando el resto de ficheros que no tienen cambios.

Se pueden recorrer todas las carpetas del árbol, y seleccionando un fichero se accede a una vista en la que se muestra el contenido de dicho fichero.

### 5.2.9. Vista contenido y diferencias de ficheros.

Se trata de dos vistas distintas pero que guardan una relación. En la primera, únicamente se muestra el contenido de un fichero en la versión seleccionada. Junto al contenido mostrado, también se ofrece una opción para poder compararlo con alguna de las versiones anteriores.

En la segunda vista se muestran las diferencias existentes entre dos versiones de un fichero. En la parte superior de esta vista se muestra un listado con todas las versiones anteriores que tienen algún cambio sobre el fichero seleccionado. Debajo de ese listado se muestra una comparativa entre las dos

versiones que intervienen, mostrando los dos ficheros, y resaltando las líneas que se han añadido, modificado o eliminado en cada una de las versiones.

### 5.2.10. Vista *wiki*.

Esta vista muestra los comentarios que los desarrolladores han publicado. Los usuarios pueden acceder a estos comentarios, y añadir o editar dichos comentarios.

En el caso de que un usuario quiere añadir nuevos comentarios, deberá acceder a la opción de edición. Desde ahí, se muestra un área de texto con el contenido del wiki para que el usuario pueda añadir nuevas líneas.

### 5.2.11. Vista contactos.

Esta vista muestra un listado con todos los contactos que están registrados en la aplicación, mostrando algunos datos del usuario: como su fotografía, su nombre y su correo electrónico. Seleccionando un contacto se accede a su perfil, y desde ahí se pueden enviar mensajes y consultar los proyectos en los que está involucrado el usuario.

### 5.2.12. Vista mensajes.

Esta vista actúa como un buzón de correo. Desde esta vista se accederá a todos los mensajes que ha recibido un usuario. Además de leer el contenido de los mensajes, desde aquí los usuarios pueden responder y eliminar los mensajes.

## 5.3. Lógica de la aplicación.

La aplicación se ha desarrollado siguiendo una programación orientada a objetos. La estructura de la aplicación viene establecida por las clases del

propio *framework*, y complementando a éstas, se utilizan clases que realizan diferentes funciones. Los objetos que se pueden crear en la aplicación hacen referencia a distintas entidades de la misma. Podemos encontrar objetos de los siguientes tipos:

- Clase "Git": los objetos de esta clase se emplean al interactuar con los proyectos alojados en el servidor.
- Clase "Rama": los objetos de esta clase se corresponden con cada una de las ramas creadas en un repositorio.
- Clase "Usuario": se utiliza para trabajar con los datos de los usuarios registrados.
- Clase "Mensaje": estos objetos son los diferentes mensajes que se envían entre los usuarios.
- Clase "Fichero", "Directorio" y "Json": estas clases son las que intervienen en la creación de los árboles de directorios.

La estructura de trabajo que incorpora el *framework* obliga a crear controladores para realizar las diferentes funciones. Existen controladores que concentran las tareas sobre un determinado sector de la aplicación:

- Controlador "Index": es el que realiza tareas de gestión en la aplicación como la presentación de contenidos y lecturas de la base de datos que no requieren de acceso al sistema de control de versiones.
- Controlador "Git": es el encargado de realizar todas las tareas que necesiten interactuar con el sistema de control de versiones.
- Controlador "Message": es el que se utiliza para realizar todas las acciones de lectura y envío de mensajes.
- Controlador "Contact": realiza todas las tareas sobre los contactos, tales como presentación del listado de nombres, y cambios en el perfil de los mismos.
- Ajax: es el encargado de realizar las tareas que se ejecutan al recibir las peticiones *http* asíncronas de los clientes.

## 5.4. Modelo de datos.

La aplicación cuenta con una conexión a una base de datos sobre la que se harán consultas para resolver funciones de la aplicación. En la figura 5.3 se detallan las entidades existentes en la base de datos. Existen dos entidades principales y otras que derivan de ellas. Las entidades con mayor peso son:

- Entidad "Usuario": en esta tabla se almacenan los datos de cada uno de los usuarios registrados en la aplicación. Cada entrada en esta tabla contiene información relativa al nombre, al correo, a la procedencia, al alias y contraseña necesarios para la autenticación. Existe un campo llamado "público" que lo que indica es si este usuario es visible al resto de usuarios. En caso de no serlo, nadie podrá enviarle mensajes privados. También se almacena el idioma que el usuario elija para mostrar el contenido. Así, siempre que acceda a la aplicación, se mostrará el contenido en el idioma escogido.
- Entidad "Proyecto": cada entrada de esta tabla hace referencia a un repositorio alojado en el servidor. De cada proyecto lo que interesa es conocer su título, cuando fue creado y el identificador de su creador. También se puede incluir una breve descripción sobre el proyecto.

De estas dos entidades derivan el resto de entidades que intervienen en la aplicación:

- Entidad "Rama": en el desarrollo de un proyecto se puede trabajar sobre diferentes ramas de trabajo. En esta tabla se almacena la información de todas las ramas que se van creando a lo largo del período de desarrollo de una aplicación. De cada rama se tiene información de su nombre, del creador y de la fecha de creación de la misma, y del repositorio en el que se encuentra.
- Entidad "Grupo": cada uno de los grupos creados en la aplicación. Tiene un campo para identificar al grupo, otro para almacenar el nombre del mismo, y otro que indica el identificar del creador del grupo.
- Entidad "Permiso": esta tabla contiene todos los permisos de acceso a los diferentes proyectos. Cuando un usuario quiere acceder a un proyecto, realiza una solicitud de acceso, insertándose una nueva entrada en

esta tabla. El administrador de cada proyecto, podrá cambiar los permisos de acceso de esas solicitudes, permitiendo o denegando el acceso, y por tanto, la posible realización de cambios sobre el mismo.

- Entidad "Permiso Grupo": en esta tabla figura el listado de proyectos a los que tienen acceso cada grupo.
- Entidad "Mensaje": esta tabla almacena todos los mensajes privados que se intercambian entre los usuarios. De cada mensaje se conoce el identificador de la persona que lo envía y del destinatario. También se guarda el asunto y el cuerpo del mensaje, así como la fecha en la que fue enviado. Existe un campo que indica el estado del mensajes, que puede ser leído o no leído, en función de si el destinatario ha accedido o no a dicho mensaje.

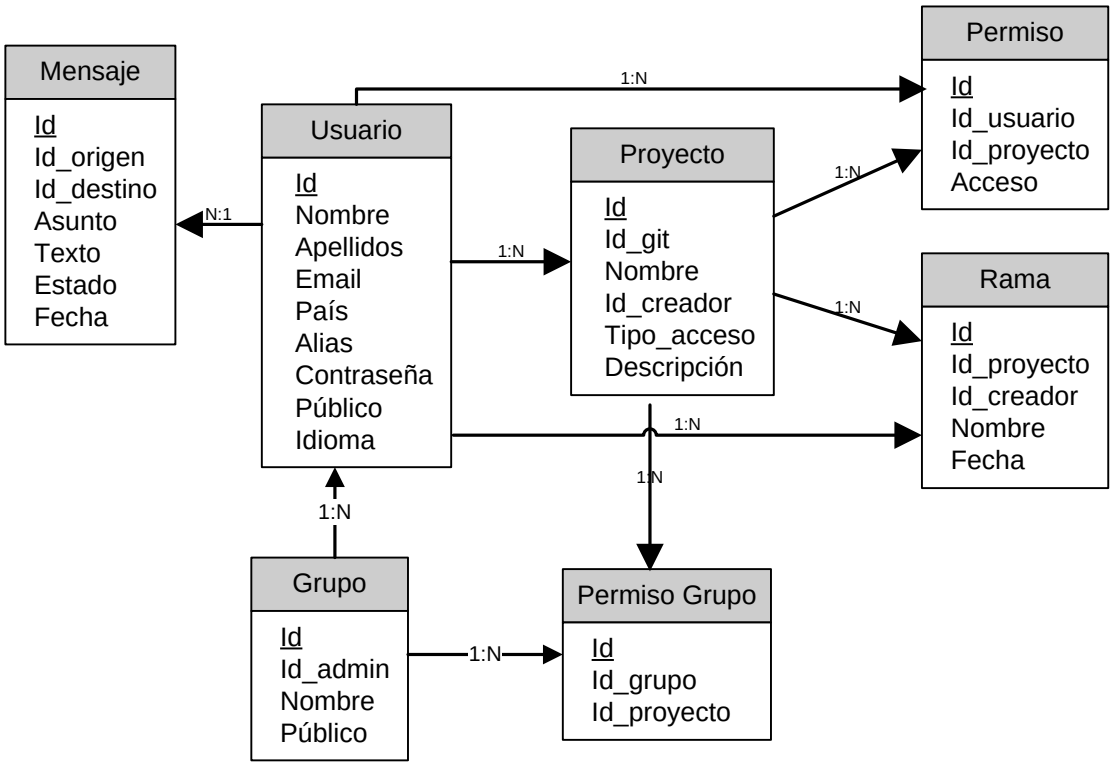


Figura 5.3: Diagrama Entidad-Relación de la base de datos

# Capítulo 6

# Implementación del sistema

En este capítulo se describen los aspectos más relevantes de la implementación del sistema, explicando las especificaciones técnicas adoptadas por la aplicación para cumplir los requisitos fijados en la misma. En el apéndice A se pueden consultar las tecnologías que intervienen en el desarrollo del proyecto.

A continuación, y siguiendo una estructura similar a la del capítulo 5, se describe como se han desarrollado los diferentes módulos integrados en la aplicación, indicando las bibliotecas de código y los *plugins* de "jQuery" utilizados para realizar la implementación.

## 6.1. Aplicación multilenguaje.

La aplicación ha sido desarrollada de tal forma que los usuarios pueden escoger el idioma en el que quieren que se presenten los contenidos. La internacionalización de la aplicación se realiza mediante un módulo del *framework* de Zend llamado "Zend Translate".

Para trabajar con este módulo existen diversos mecanismos como el uso de un *array* con las traducciones o ficheros con formato "csv", pero el que la aplicación utiliza es "Gettext", que es la biblioteca de GNU para internacionalización. Esta biblioteca trabaja con ficheros ".po" (*Portable Object*) y ".mo"(*Machine Object)*. Para ello se crea un fichero de texto ".po" por cada idioma. Cada uno de estos ficheros contiene pares formados por un identifi-

cador (*msgid*) y el texto que aparecerá en sustitución de éste (*msgstr*). Estos ficheros se compilan para generar los ficheros binarios ".mo", que son los que utilizará la aplicación para la lectura de los textos.

Para que este mecanismo de internacionalización pueda trabajar con los ficheros de las traducciones necesita que éstos estén almacenados en el servidor siguiendo una determinada estructura de directorios. En la figura 6.1 se muestra esta estructura de directorios.

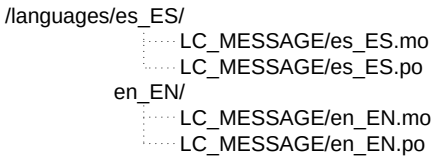


Figura 6.1: Estructura de los directorios y ficheros de internacionalización.

A la hora de presentar los diferentes contenidos de la aplicación Web es necesario configurar previamente el idioma. Para ello existen funciones, en el propio *framework*, mediante las que se gestiona el idioma activo (*locale*). En el menú superior de cada vista se puede seleccionar el idioma para mostrar los contenidos. Cuando un usuario selecciona uno de los idiomas, el sistema guarda en la base de datos el idioma seleccionado. Así, cada vez que un usuario accede a la aplicación, se presenta el contenido en el idioma que éste hubiese seleccionado la última vez. Este idioma está almacenado en la base de datos, junto al resto de información del usuario.

En la figura 6.2 se ilustra el flujo de trabajo que tiene la aplicación a la hora de realizar la traducción de los textos. Una vez cargado y configurado el idioma con el que se mostrará el contenido, la aplicación accede al fichero que contiene las traducciones de dicho idioma, y sustituye cada identificador por la traducción que tenga asociada. El proyecto ofrece la posibilidad de mostrar los contenidos en castellano o en inglés. Pero gracias a esta forma de implementación añadir nuevos idiomas es una tarea sencilla, ya que sólo hay que coger uno de los fichero ".po" ya creados y traducir los textos que aparezcan en él.

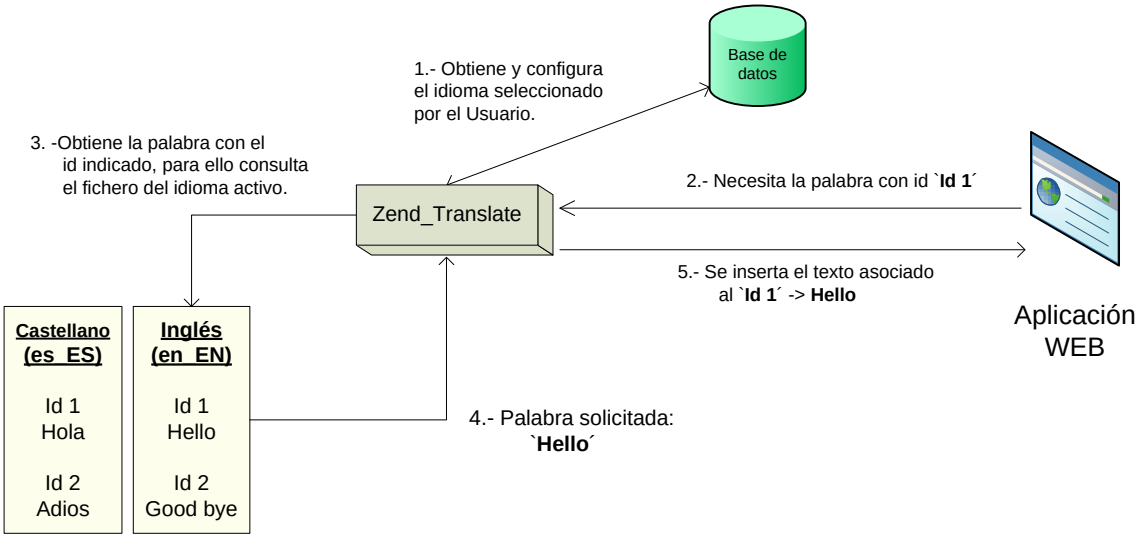


Figura 6.2: Aplicación multilenguaje, lectura de ficheros de traducciones.

## 6.2. Conexión entre PHP y Git.

Esta conexión es el núcleo de la aplicación. Todas las tareas sobre los repositorios se realizarán mediante la comunicación entre PHP y el sistema de control de versiones. Como se ilustra en la figura 6.3, existen dos vías para poder realizar la comunicación entre ellos. El utilizar una u otra dependerá del tipo de tarea que se esté ejecutando, si es de lectura o de escritura.

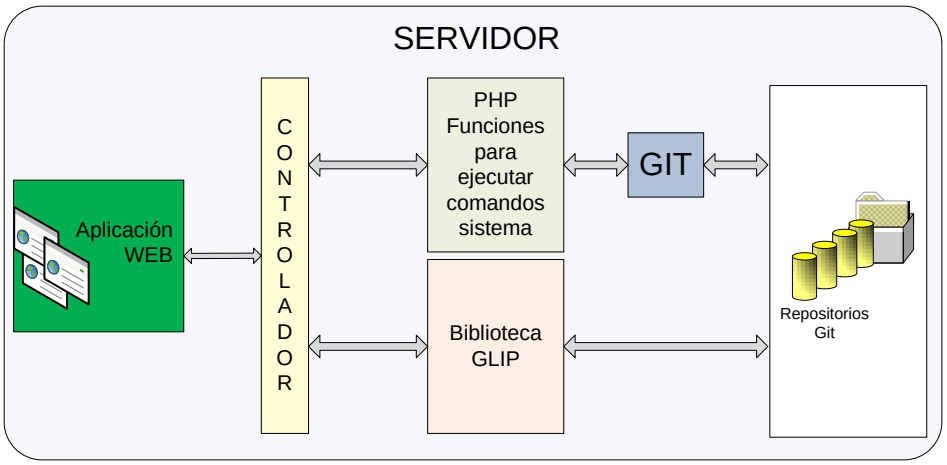


Figura 6.3: Conexión entre Git y PHP.

### 6.2.1. Glip (*Git Library in PHP*).

Para trabajar con Git desde PHP existe una biblioteca de código conocida como Glip [4]. Gracias a esta biblioteca se pueden realizar tareas de lectura en los repositorios alojados en el servidor. Esta biblioteca trabaja con objetos, del tipo "Gitcommit", "Gitobject" y "Git". Estos objetos se corresponden con las versiones, los ficheros y los repositorios con los que trabaja Git.

Para realizar una consulta, únicamente hay que crear un objeto que haga referencia al repositorio sobre el que se realizará. *Glip* accede a los objetos con los que trabaja Git, generando posteriormente un objeto, del tipo *Gitcommit* o *Gitobject*. Estos objetos tienen como atributos la información relativa a un *commit* o a un fichero.

Gracias a esta biblioteca se puede acceder al contenido de cualquier fichero o al listado de los ficheros existentes en una determinada versión. Para ello es necesario el identificador *hash* con el que trabaja Git para diferenciar las versiones y ficheros. Tras realizar la consulta, *Glip* devuelve en un *array* los datos solicitados, con la información parseada de las versiones o de los ficheros.

También se utiliza esta biblioteca para obtener los ficheros que han sido modificados entre dos versiones. Indicando el *hash* de cada una de las versiones, esta biblioteca devuelve un *array* formado por los nombres de los ficheros que han sido modificados, y si éstos han sido eliminados, añadidos o simplemente modificados.

Esta biblioteca también ofrece ciertas funciones de escritura, pero se ha optado por realizar estas tareas directamente sobre el sistema de control de versiones, ya que éste presenta más opciones y funcionalidad que la implementada por la biblioteca.

### 6.2.2. Llamadas por líneas de comandos.

Las tareas de escritura, y alguna de lectura, hay que realizarlas directamente mediante comandos de Git. En una máquina local ésto se realizaría por consola, introduciendo los comandos con los que trabaja Git. En el servidor, se realiza del mismo modo, hay que hacer las llamadas a Git por línea de

comandos. Para poder realizar ésto, PHP dispone de funciones para ejecutar comandos del sistema, como la función "*shell_exec*". Estas funciones reciben como parámetro un comando, lo ejecutan y devuelven la salida generada por la ejecución de dicho comando.

En el proyecto se ha creado una clase que contiene todas las tareas que se necesitan realizar sobre los repositorios a través de Git. Estas funciones efectúan las llamadas a Git, y almacenan la salida que éste genera. Luego se analiza el contenido de la respuesta almacenada, y se genera un *array* con la información ya tratada para que la aplicación pueda realizar todas las tareas requeridas en cada momento.

## 6.3. Control de acceso.

### 6.3.1. Control de acceso a la aplicación Web.

La aplicación cuenta con un sistema para controlar el acceso de los usuarios a la misma, por lo que sólo los usuarios registrados podrán acceder. El registro se realiza desde la vista inicial, rellenando el formulario de acceso para nuevos usuarios. Cada usuario registrado es insertado en la base de datos como una nueva fila.

Cada vez que un usuario accede a la aplicación debe introducir su alias y su contraseña. Cuando el servidor recibe la petición de acceso, se comprueba si existe alguna entrada en la base de datos con esa información. En caso afirmativo, la autenticación es correcta, permitiéndose el acceso a la aplicación y creándose una nueva sesión; en caso contrario, se deniega el acceso y se redirige la aplicación de nuevo al formulario de acceso.

### 6.3.2. Control de acceso a los repositorios.

La aplicación también controla el acceso a cada uno de los repositorios alojados en el servidor. Los usuarios sólo deben poder realizar cambios sobre los repositorios que han creado ellos, o sobre los repositorios a los que tengan permisos de acceso.

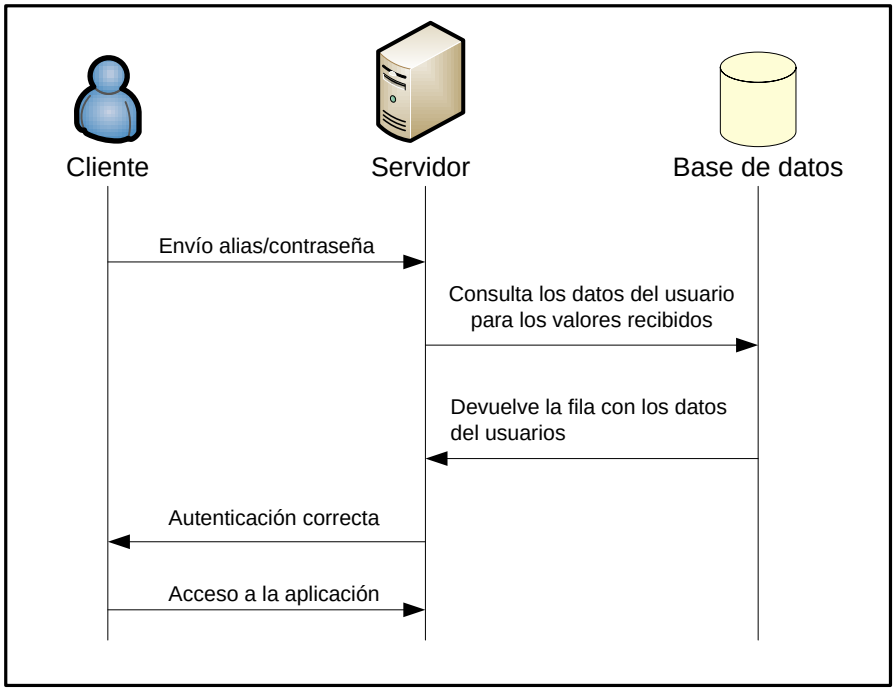


Figura 6.4: Autenticación de usuario correcta

Para controlar ésto, en cada repositorio se definen tres roles con unos permisos determinados:

1. Administrador: es el usuario que creó el repositorio. Dicho administrador es el único que puede permitir o denegar el acceso al resto de usuarios. Es el encargado de las tareas de gestión sobre el repositorio.

2. Miembro de un grupo: cualquier usuario que pertenezca a un grupo podrá acceder y realizar cambios sobre los proyectos en los que trabaje dicho grupo. En este caso, los permisos son concedidos a un grupo, y estos permisos son a su vez concedidos a todos los miembros de dicho grupo.

3. Usuario: cualquiera que utilice la aplicación, sin necesidad de pertenecer a un grupo, puede solicitar permisos de acceso a un proyecto. Los permisos se le conceden directamente a dicho usuario.

En el listado de los proyectos, presentado en la vista inicial, un usuario puede ver los proyectos a los que tiene acceso, y solicitar el acceso a cualquier

repositorio con acceso bloqueado. Cuando un usuario solicita los permisos de acceso a un repositorio, se crea una solicitud que se almacena en la base de datos como una nueva entrada. Esta solicitud relaciona a un usuario con un determinado repositorio, indicando en el estado de dicha solicitud que se encuentra pendiente de confirmación. El administrador de cada repositorio tiene acceso a un listado que muestra todas las solicitudes recibidas para un determinado repositorio. Desde este listado puede cambiar los permisos de cualquiera de los usuarios o grupos, permitiendo o bloqueando el acceso. El permitir a un usuario o grupo acceder a un proyecto implica que se le conceden permisos para realizar cualquier tipo de tarea de lectura o escritura.

En el caso de los grupos, el comportamiento es similar. El creador de un grupo será el encargado de solicitar los permisos de acceso para dicho grupo. Si se permite el acceso a un grupo, todos los usuarios que pertenezcan a ese grupo podrán realizar cambios sobre el repositorio.

## 6.4. Gestión de los repositorios.

En esta sección se habla de diversas medidas que se han tomado a la hora de trabajar sobre los repositorios. Al principio se trabajó directamente sobre el repositorio original, pero finalmente, se optó por tomar otras medidas para resolver una serie de problemas. Estas medidas se describen en los siguientes puntos.

### 6.4.1. Configuración de los repositorios.

Al iniciar un repositorio se cambian algunos valores en su configuración. La aplicación asigna el valor "*true*" al parámetro de configuración "*core.bare*". Con ésto lo que se consigue es que el repositorio creado no tenga asociado ningún directorio de trabajo, por lo que no se podrán realizar ciertas tareas directamente sobre él. Este cambio también produce que se deshabiliten comandos de Git como "git add" y "git merge". Por tanto, ahora para poder realizar cambios sobre los repositorios hay que hacerlo desde una ubicación externa a éstos.

### 6.4.2. Clonado de los repositorios remotos.

La solución adoptada para trabajar sobre los repositorios fue la de clonar cada uno en otra ubicación dentro del servidor. Estos clones no están configurados como repositorios de tipo "*bare*", por lo que si se pueden realizar las tareas necesarias para la creación de nuevas versiones sobre ellos.

Existe un clon por cada rama del repositorio, cada vez que se crea una rama también se crea su correspondiente clon. Estos clones tienen en su fichero de configuración la ubicación del repositorio remoto con el que deben mantener la sincronización. Sobre estos clones se descomprimen los ficheros que se reciben con los nuevos cambios, y se ejecutarán todas las tareas que intervienen en la realización de los *commits*. Cada vez que se genera una nueva versión, o cambios que deban ser actualizados en la rama remota, se debe hacer una llamada al comando "*git push*". Este comando se encarga de mantener sincronizados la rama remota con su clon correspondiente, enviando a la rama remota los cambios que se hayan realizado sobre la clonada.

Gracias a clonar las ramas de cada repositorio y trabajar sobre ellas se consigue que en el servidor siempre haya una versión estable de los datos de un proyecto. Esto es así porque sobre el repositorio remoto no se pueden realizar acciones que puedan comprometer el funcionamiento de la aplicación, ya que todas las acciones sobre ellos las ejecuta Git y no nuestra aplicación. También gracias a ésto se consigue reducir el tiempo de ejecución de las diferentes tareas, ahora no es necesario clonar una rama cada vez que se quiera realizar algún cambio sobre ella.

### 6.4.3. Control de acceso concurrente a los repositorios.

En la aplicación ha sido necesario implementar algún mecanismo para asegurar que dos usuarios no pueden realizar cambios sobre una rama al mismo tiempo. Hay acciones como la de realizar un nuevo *commit* que si dos usuarios la realizasen a la vez sobre la misma rama provocaría un "conflicto" entre ambos, ya que la aplicación no sabría los ficheros que ha enviado uno u otro.

Para solucionar ésto se ha implementado una mecanismo que consiste en crear un fichero de texto cada vez que un usuario esté realizando cambios so-

bre una rama. Estos ficheros se utilizan como un bloqueo. Cuando un usuario inicia las acciones para generar un nuevo *commit* se crea este fichero. Si otro usuario intenta acceder en ese instante se le muestra un mensaje indicando que debe esperar a que termine el usuario anterior. El bloqueo de la rama durará hasta que el usuario finalice las tareas que esté realizando, o hasta que se pase un limite de tiempo configurado, tras el cual, el bloqueo desaparecería y las acciones sin terminar se cancelarían. Con el temporizador se evita que un usuario pueda dejar sin finalizar un *commit*, se le da un tiempo suficiente para realizarlo, y si en ese tiempo no se ha confirmado entonces se cancela.

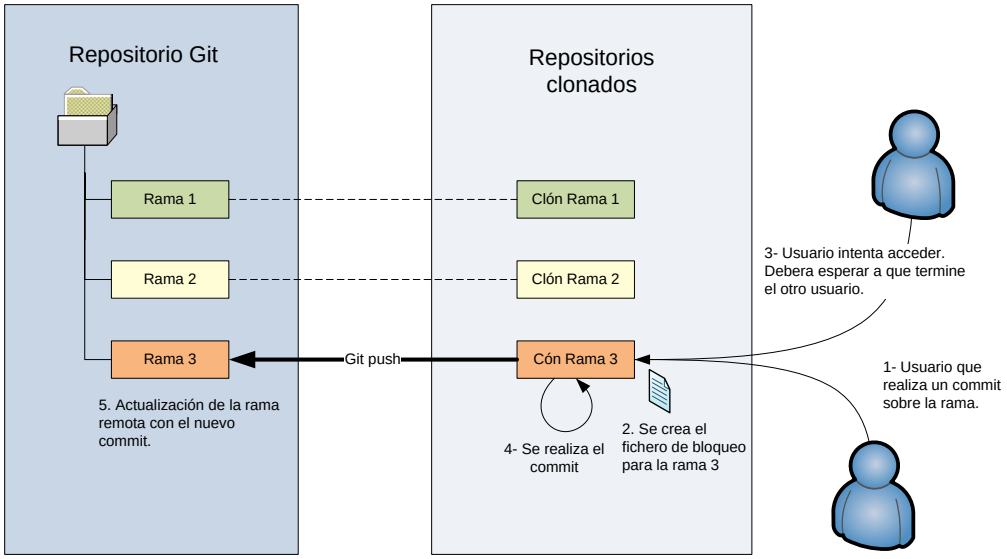


Figura 6.5: Commit sobre rama clonada y actualización en la remota.

### 6.4.4. Mantenimiento de los clones.

En los clones de los repositorios es donde se realizaran todos los cambios. Cuando un usuario esté realizando un nuevo *commit* o un nuevo *merge* se tienen que hacer cambios sobre éstos. Estas acciones cambian el estado en el que se encuentran los directorios de trabajo de los clones. Por esta razón, el primer paso para realizar cualquier acción sobre los clones es dejar el directorio de trabajo en el mismo estado en el que estaba el la última versión almacenada. Con ésto se evita que otros usuarios hayan dejado algún fichero en este directorio en un *commit* anterior sin finalizar. Después de restaurar este estado, ya se realizan las tareas pertinentes.

## 6.5. Trabajo con ficheros comprimidos.

La aplicación tiene que trabajar con ficheros comprimidos para crear nuevos proyectos, para realizar nuevos *commits* y para que los usuarios puedan descargar el contenido de los proyectos. Para cada una de estas tareas se emplean diferentes mecanismos. A continuación se detallan los modos de trabajo con este tipo de ficheros, en el caso de que se reciban en el servidor, y en el caso de que sea éste el que los tiene que generar.

### 6.5.1. Envío de ficheros al servidor.

Hay dos instantes en los que es necesario subir ficheros al servidor, uno es cuando se crea un nuevo proyecto, y el otro es cuando se está realizando un nuevo *commit*. Los ficheros se envían de manera diferente en función de las tareas que se tengan que ejecutar en el servidor.

Cuando se crea un nuevo repositorio, únicamente hay que descomprimir el fichero que se reciba e iniciar dicho repositorio. Estas tareas no tienen una duración muy elevada para la aplicación. En cambio, cuando se realiza un *commit*, hay que realizar varias acciones cuya duración se debe tener en cuenta. Al crear una nueva versión hay que descomprimir el fichero recibido en el servidor y obtener el listado de todos los ficheros modificados, añadidos o eliminados. Todas estas acciones se podrían realizar tras recibir la petición *http* con el formulario, pero entonces habría que sumar los tiempos de ejecución de cada una de ellas y la aplicación quedaría "bloqueada" hasta que se finalizasen todas la tareas en el servidor. Por esta razón, para tratar de suavizar la espera del usuario, la subida de los ficheros se realiza mediante una petición *http* asíncrona, de tal manera que el usuario seguirá rellenando el formulario con los datos de la nueva versión mientras que en el servidor se prepara el repositorio y se descomprime el fichero comprimido.

Por tanto, en la aplicación se han implementado dos mecanismos para realizar el envío de ficheros al servidor:

1. Para crear un nuevo repositorio, el usuario debe rellenar un formulario con los datos del mismo. En este formulario, además de campos de texto, se añade un campo de tipo "archivo". Este será el fichero com-

primido que contiene los archivos del repositorio que se va a crear. El usuario envía al servidor el formulario en una petición *http*, y el servidor recoge todos los argumentos que recibe en esa petición. Esta es la forma común de subir ficheros a un servidor mediante una petición *http*. Una vez recibida la petición, se almacena el fichero comprimido en un directorio del servidor hasta que se realicen todas las tareas pertinentes con él.

2. A la hora de realizar un nuevo *commit*, el fichero comprimido se envía al servidor antes de que se envíe el formulario. Para conseguir ésto se utiliza un *plugin* de jQuery llamado "uploadfy" [41], que realiza una petición *ajax* asíncrona, en la que se envía el fichero comprimido al servidor. Este *plugin* utiliza Flash y JavaScript para realizar el envío. Permite realizar la subida múltiple de ficheros, aunque en el proyecto se han configurado ciertos parámetros del *plugin*: sólo se permite seleccionar ficheros con formato "Zip" o "Rar", el envío del fichero se realiza automáticamente en cuanto el usuario selecciona el fichero comprimido, y también se le indica el nombre del *script* en el servidor que recibirá y tratará el fichero. Cuando el servidor recibe el fichero, realiza las tareas programadas en dicho *script*.

### 6.5.2. Comprimir y descomprimir ficheros.

La aplicación trabaja con ficheros comprimidos cuando se crea un nuevo proyecto, cuando se realiza un nuevo *commit*, y cuando un usuario quiere descargar el contenido de alguna versión. Estas acciones implican tareas de compresión y descompresión de ficheros, que se realizan de la siguiente manera:

- Para descomprimir ficheros en el servidor, PHP tiene una serie de bibliotecas de código que se pueden utilizar, como "pclzip" [15]. A la hora de escoger entre estas bibliotecas hay que tener en cuenta la plataforma sobre la que se desarrolla la aplicación. Este proyecto está desarrollado en "Windows", y dichas bibliotecas presentan problemas con la codificación, y extraen los ficheros con un nombre mal codificado. Por tanto, se ha optado por utilizar la herramienta "unzip" para descomprimir los ficheros con extensión "Zip", y "unrar" para descomprimir los ficheros con extensión "Rar". Gracias a utilizar estas herramientas se consigue

que la aplicación sea portable a una plataforma "Unix", ya que estas herramientas funcionan correctamente también en dicha plataforma.

- La aplicación necesita crear un fichero comprimido cuando se quiera descargar el contenido de una versión determinada de un proyecto. Esta acción se realiza directamente mediante el comando "*git archive*", que es un comando propio del sistema de control de versiones. Este comando genera un fichero comprimido con sólo indicar el *hash* de la versión o del directorio que se quiera comprimir. Esta función permite escoger la extensión del fichero que se va a generar, *zip* o *tar*. Por tanto, para comprimir ficheros, únicamente hay que hacer una llamada por línea de comandos a Git.

## 6.6. Generar árbol de directorios.

Para facilitar el trabajo de gestión de los distintos repositorios, los usuarios deben poder moverse por las diferentes carpetas que contiene cada una de las versiones, y así poder consultar el contenido de los ficheros y los cambios que se han producido en ellos. En la aplicación hay dos instantes en los que es necesario presentar por pantalla un listado de todos los directorios y ficheros contenidos en una determinada versión de un proyecto. Esto se produce en el momento de realizar un nuevo *commit*, donde es necesario mostrar todos los ficheros nuevos, modificados o eliminados para que el usuario escoja los cambios a guardar; y el otro instante es cuando se consultan los archivos contenidos en alguna de las versiones almacenadas.

Para generar el árbol, la aplicación utiliza un *plugin* de jQuery llamado "jsTree" [12], distribuido bajo licencia GPL. Gracias a este *plugin* se puede generar un árbol de directorios y aplicarle diferentes estilos. Para poder generar el árbol hay que indicarle al *plugin* los datos que se van a representar. El *plugin* trabaja con datos en estos formatos: HTML, XML o JSON. En la aplicación se ha optado por trabajar con datos JSON.

A la hora de generar el árbol es necesario realizar ciertas tareas, tanto en el servidor como en el navegador del cliente. A continuación se indican los pasos que se siguen para la generación del árbol:

1. Primero se obtiene el listado de directorios y ficheros. Para ello en el

servidor se hace una consulta a Git, utilizando la función "*listRecursive*" sobre un objeto del tipo "GitTree". Esta función pertenece a la biblioteca *Glip*. A partir del *hash*, que identifica la versión, la función devuelve un *array* en el que cada posición se corresponde con el nombre y el *path* de cada fichero o directorio.

2. Se analiza la información contenida en el *array* obtenido. Para realizar esta tarea se trabaja utilizando objetos, del tipo directorio o del tipo fichero. Se recorre el *array* y se crean objetos del tipo especifico. Los objetos del tipo directorio tienen como atributos el nombre del directorio y un *array* con todos los ficheros y subdirectorios que contiene. Los ficheros tienen como atributo el nombre y el estado del fichero, que en el caso de estar realizando un *commit* puede ser creado, eliminado o modificado.

3. Una vez creados todos los objetos, se tiene que generar la respuesta con la información necesaria para que el *script* que se ejecuta en el cliente pueda crear el *json*. El *plugin* tiene que recibir como parámetro una cadena de texto que contenga la estructura que se quiere cargar en el *json*. Esta estructura tiene un formato definido en la documentación del *plugin*. Por tanto, la aplicación genera en el servidor únicamente la cadena de texto con el formato requerido que se enviará en la respuesta, no se crea ningún objeto *json* en el servidor.

4. Se envía la respuesta al cliente, incluyendo en ella la cadena creada y los parámetros de configuración para el *script*. Cuando se carga la página en el navegador del cliente se ejecuta un *script* que es el que muestra el árbol con el estilo configurado. Este *script*, a partir de la cadena de texto que recibe en la respuesta, carga el árbol *json* que necesita para poder trabajar con el árbol de directorios.

Para disminuir el tiempo de ejecución de esta función en el servidor, la aplicación almacena en ficheros de textos la cadena *json* que se crea. Se crean dos ficheros, en uno almacena la cadena con el árbol completo, y en otro la cadena con el nombre de los ficheros que han sido modificados en una versión. Estos ficheros se identifican con el *hash* de la versión del *json* que estén almacenando. Estos ficheros se guardan en una carpeta distinta para cada proyecto y rama, y gracias a nombrarlos utilizando el *hash* de la versión se evitan problemas al no poderse repetir el nombre de estos ficheros.

Estos ficheros actúan como una *caché*, y la aplicación sólo realizará las funciones necesarias para la creación del *json* y estos ficheros una sola vez

por cada versión que se guarde en el repositorio. Gracias a ésto se consigue disminuir el retardo en la aplicación, ya que se evita tener que procesar toda esta funcionalidad cada vez que se accede a una versión.

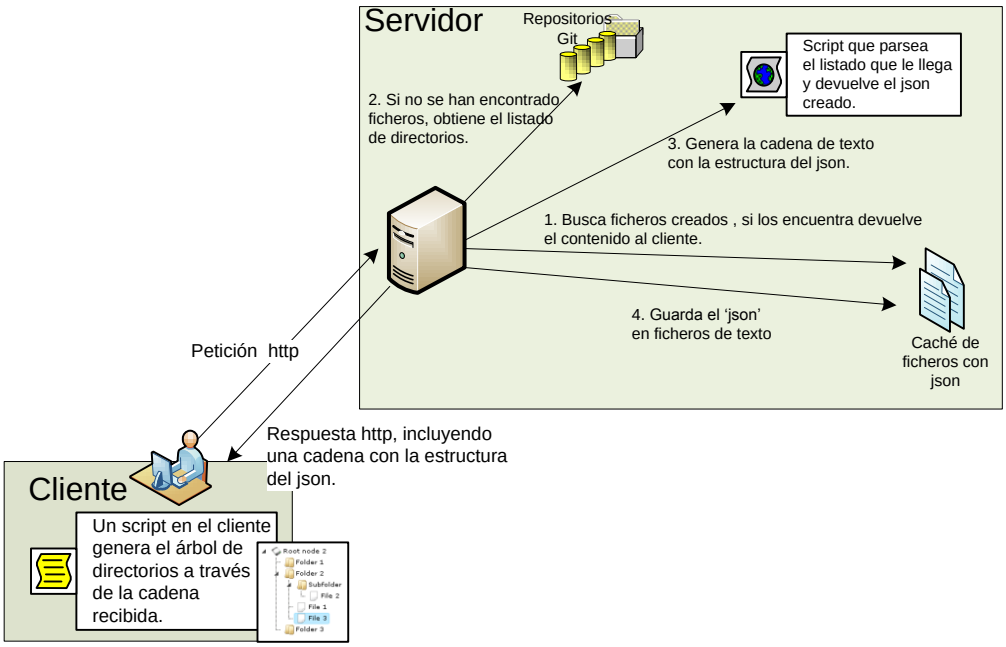


Figura 6.6: Creación del árbol de directorios.

## 6.7. Mostrar el contenido y las diferencias entre dos versiones de un fichero.

### 6.7.1. Contenido de un fichero.

En la aplicación se puedo acceder al contenido que tenía un fichero en una determinada versión. Para ello se selecciona la versión y el fichero que se quiera consultar. Mediante la librería "glip" se accede al contenido del fichero. Se crea un objeto a partir del *hash* del fichero, que tendrá en el atributo *data* el contenido del fichero.

Para mostrar el fichero se utiliza una biblioteca llamada "GeSHi (*Generic Syntax Highlighter*)" [10] que muestra el contenido del fichero utilizando

diferentes colores en función de la extensión del mismo. Para mostrar el contenido únicamente hay que pasarle a "GeSHi" dos parámetros, uno con el contenido del fichero y otro con la extensión del mismo, y luego llamar a la función "*parseCode*" que es la encargada de mostrar el texto en diferentes colores.

### 6.7.2. Diferencias entre dos versiones de un fichero.

Uno de las principales tareas que debe realizar un sistema de control de versiones es mostrar los cambios realizados en un fichero entre cada una de las versiones del proyecto. En la aplicación, existe una acción que permite comprobar estos cambios. Para ello, se compara la última versión existente con cualquiera de las anteriores.

Esta tarea se realiza en varias fases. La primera es obtener un listado con las versiones anteriores. Como todas las tareas de lectura de un repositorio, se pueden realizar mediante la biblioteca *glip* o mediante consultas sobre Git. En esta caso, es necesario hacerlo directamente sobre Git, ya que necesitamos conocer únicamente el listado de las versiones que tienen algún cambio en el fichero indicado. Si se utilizasen las funciones de *glip* se obtendrían todas las versiones, tengan o no cambios sobre dicho fichero. Por tanto, utilizando las opciones del comando "git log" de Git, se genera un *array* con el nombre e *hash* de cada versión en la que se haya modificado el fichero.

El usuario puede seleccionar cualquiera de las versiones obtenidas en el paso anterior para mostrar las diferencias. Una vez seleccionada la versión, con la que se realizará la comparación, es necesario obtener el contenido del fichero tanto en la versión actual como en la versión que se haya seleccionado. La versión actual se obtiene a través de la biblioteca *glip*, mediante una función que recibe como parámetro el *hash* del fichero y devuelve el contenido del mismo. Para obtener el contenido almacenado en una versión anterior es necesario hacer una llamada a Git por línea de comandos. Mediante el comando *show* de Git, indicándole el nombre del fichero y el *hash* de la versión seleccionada, se obtiene el contenido que tenía en dicho instante.

Finalmente, una vez obtenido el contenido del fichero en cada versión, se utiliza un *plugin* de *javascript*, llamado "jsdiff" [31] que muestra las diferencias entra ambas versiones. Para ello, el *plugin* recibe el contenido del fichero en cada uno de los instantes, y realiza las comprobaciones necesarias para

obtener todas las diferencias entre ellos. Una vez obtenidas se configura el estilo con el que se mostrarán los ficheros. El estilo escogido muestra las dos versiones una al lado de la otra, resaltando con diferentes colores las líneas que se han modificado, añadido o eliminado en cada versión.

## 6.8. Presentación del contenido en tablas.

Para presentar contenido en tablas se utiliza un *plugin* de jQuery llamado "DataTables" [34]. Este *plugin* permite aplicar diferentes estilos a las tablas HTML, y también les añade nueva funcionalidad:

- Limitar el número de resultados mostrados por página. En función del número de resultados que se muestren, el *plugin* calcula el número de páginas que tiene, ocultando o mostrando los resultados en función de la página que el usuario seleccione en cada momento.
- Filtrado de contenido por filas: permite introducir una palabra en un campo de texto, y el *plugin* se encarga de mostrar únicamente las filas que contienen palabras con esas coincidencia.
- Permite ordenar el contenido de la tabla por columnas. Seleccionando una columna puede ordenarse su contenido por orden alfabético, de modo ascendente o descendente.

Para trabajar con este tipo de tablas, únicamente hay que generar la tabla HTML siguiendo una estructura determinada, distinguiendo la cabecera y el cuerpo. Luego se envía la respuesta *http* al cliente. En el navegador del cliente se muestra la página Web con la tabla creada. Una vez cargada la página, se ejecuta el *script* que se encarga de darle el estilo y la funcionalidad. A este *script* se le pasan distintos parámetros de configuración para seleccionar diferentes estilos, restricciones de ordenación, y para habilitar o deshabilitar ciertas funciones.

## 6.9. Intercambio de mensajes entre usuarios.

A la hora de desarrollar un proyecto, es importante que las personas que estén trabajando sobre él estén en contacto. Por ello, la aplicación cuenta con un mecanismo de comunicación entre usuarios. Se ha optado por implementar un sistema de mensajería que trabaje sobre la base de datos, ya que no implica mucho tiempo su desarrollo y cumple con los requisitos de la aplicación. En este sistema de comunicación intervienen dos entidades: los usuarios y los mensajes.

Los usuarios pueden acceder a una vista en la que se presenta un listado con el nombre del resto de usuarios registrados en la aplicación. Para que un usuarios aparezca en esta lista debe tener su perfil como público, sino el resto de usuarios no pueden comunicarse con él. Desde esta vista, cualquier usuario puede seleccionar a otro para enviarle un mensaje privado.

Cada mensaje que se envía se almacena como una entrada en la base de datos. Cuando se envía un nuevo mensaje, se inserta una nueva fila en la tabla de mensajes. Esta entrada contiene el identificador del usuario que lo envía y del usuario destinatario, además del estado del mensaje, leído o no leído. Los mensajes se almacenan en la base de datos hasta que los usuarios decidan borrarlos. Los usuarios pueden leer, responder o eliminar cualquiera de los mensajes que han recibido. En la figura 6.7 se observa primero el envío de un mensaje, desde la recepción en el servidor hasta que se almacena en la base de datos; y por otro lado, se observan los pasos que se llevan a cabo cuando un usuario consulta los mensajes que ha recibido.

## 6.10. Wiki para compartir comentarios y documentación.

Para que los desarrolladores puedan compartir sus ideas y documentación, además del sistema de envío de mensajes implementado, se ha incluido un *wiki*. Se ha optado por hacer un *wiki* basado en ficheros de texto, ya que por razones de tiempo ésta es la medida más rápida de implementar e integrar.

Para cada proyecto existirá un *wiki* general, en el que se añadirán los

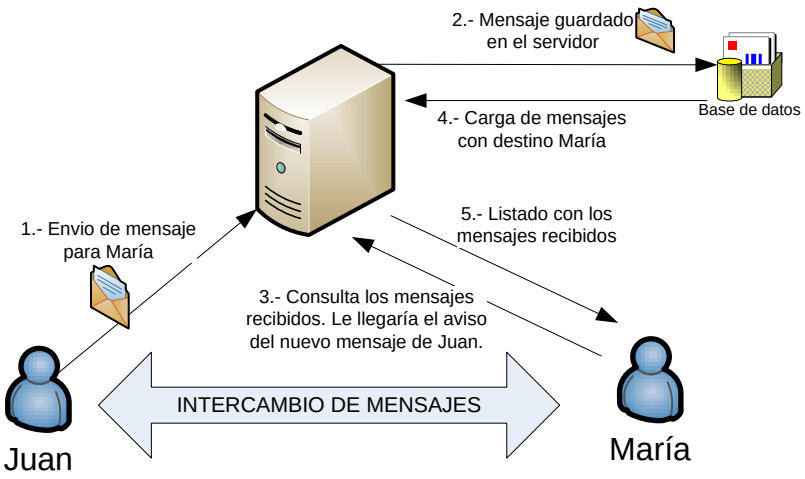


Figura 6.7: Intercambio de mensajes entre usuarios.

comentarios generales para dicho proyecto. También existirá un *wiki* por cada rama creada en el repositorio. El *wiki* no es más que un fichero de texto en el que se guarda el contenido que los usuarios vayan editando.

En una de las vistas de la aplicación se muestra el contenido de estos ficheros con las aportaciones que se hayan introducido. Si algún usuario quiere editarlo, se muestra un área de texto con dicho contenido, y se permite que se añadan cambios. Cuando se haya finalizado la edición, se sobrescribe el fichero de texto del *wiki* con el contenido del área de texto que se ha editado.

# Capítulo 7

# Pruebas

Para realizar un chequeo del correcto funcionamiento de una aplicación software se deben llevar a cabo diversas pruebas. Las primeras pruebas realizadas tenían como objetivo comprobar el correcto funcionamiento de todos los módulos de la aplicación. Para ello se prueban todas las acciones que la aplicación debería realizar. A continuación se presenta un listado con las pruebas que la aplicación ha pasado con éxito:

- Creación de nuevos repositorios. Esta prueba ha generado problemas de compatibilidad entre los navegadores. Al acceder al contenido del *textarea* que tiene el texto del resumen del nuevo *commit* en "Internet Explorer" no funcionaba utilizando el mismo método "val()" que en los otros navegadores. Estas funciones utilizadas son de la biblioteca *jQuery*. Por eso se tiene que detectar el navegador, y luego en función de si es uno u otro utilizar la función "val()" o "text()" para extraer el contenido de este campo.
- Creación de nuevos *commits*.
- Creación de nuevas ramas de trabajo.
- Alta de nuevos usuarios.
- Creación de nuevos grupos.
- Envío y recepción de mensajes entre usuarios.
- Solicitud de acceso a un determinado proyecto.

- Cambios en los permisos de acceso a un proyecto realizados como administrador del mismo.
- Incluir comentarios en la wiki de los proyectos.
- Acceder a los ficheros contenidos en una versión concreta.
- Acceder al contenido de un fichero dentro de una versión concreta.
- Mostrar las diferencias entre versiones.
- Comparar dos versiones de un mismo fichero.
- Combinar dos ramas.
- Descargar en un fichero comprimido cualquier versión de un proyecto.
- Realizar un *checkout* de una rama para crear una nueva.
- Gestión de los grupos, añadir y eliminar usuarios de un grupo.
- Cambiar el idioma con el que se presentan los contenidos en la aplicación.
- Navegación entre las vistas.

Las pruebas anteriores no se centran en buscar errores específicos sobre las funciones, simplemente se encargan de comprobar que la aplicación realice todas las acciones sin que se produzcan errores.

Complementando a estas pruebas, se han realizado otras que se centraban en testear diversas funciones que resultan críticas en el funcionamiento de la aplicación. Entre estas funciones destacan las que realizan acciones de lectura o escritura en los repositorios como las siguientes:

- Se han realizado pruebas para descartar problemas en la codificación de los caracteres. Esta prueba consistía en crear proyectos, realizar *commits* y crear ramas utilizando caracteres que no fuesen ASCII. Al realizar las pruebas se detectaron diversos problemas de codificación que obligaron a realizar cambios en la aplicación.

  El primer problema surge por las bibliotecas de código que se utilizaban para descomprimir los ficheros en el servidor. Estas bibliotecas trabajan con caracteres ASCII. La prueba realizada consistía en introducir

alguna tilde o algún carácter especial en los nombres de los archivos que se incluían el fichero comprimido. Las funciones que extraían estos archivos en el servidor cambiaban los caracteres que no fuesen ASCII del nombre por otros, por lo que extraía los ficheros con un nombre mal codificado. Para solucionar ésto se escogió otro mecanismo para descomprimir los ficheros, explicado en la sección 6.5.2. Gracias a este mecanismo se solucionaron los problemas con la codificación en los ficheros comprimidos.

El segundo problema apareció a la hora de consultar el estado de un repositorio, con el comando "*git status*". La salida que se genera viene codificada con caracteres ASCII, y si aparece algún carácter no imprimible, devuelve una cadena de texto rodeada por comillas dobles, y con los caracteres no imprimibles representados en base octal. Esto provocaba problemas en la aplicación, al aparecer las comillas y el texto con caracteres escapados, fallaba el *script* encargado de crear el árbol de directorios y también fallaba al intentar acceder al contenido de un fichero ya que el nombre que se le pasaba no se correspondía con ninguno de los que había en el repositorio. Para solucionar ésto, tras probar varias medidas, se empleó una función de PHP llamada "*stripcslashes*", que devuelve una cadena de texto cambiando los caracteres escapados por su carácter correspondiente.

Con estas dos medidas se solucionaron los problemas de codificación de los caracteres.

- Para evitar problemas con los caracteres que un usuario pueda introducir en el formulario a la hora de crear nuevos proyectos y *commits*, se realizan comprobaciones en dichos formularios antes de ser enviados al servidor. Esto obliga al usuario a utilizar únicamente caracteres alfanuméricos, evitando así que se generen errores por los caracteres utilizados. Como todas las comprobaciones que se realizan en el navegador del cliente mediante *javascript*, también se comprueban los datos introducidos de nuevo en el *script* que los recibe en el servidor.

- Problemas cuando se accedía al contenido de un fichero cuando no se sabe que extensión tiene. Si no se indica la extensión del fichero la aplicación genera un error. Se ha creado una función que extrae la extensión del fichero para que se muestre correctamente.

- Cuando se realiza la acción de combinar dos ramas es necesario que esté configurado el parámetro "*global-user*" para que se realice correctamente.

- Guardar una nueva versión detectando ficheros eliminados. Para ello el usuario deberá haber seleccionado estos ficheros en el árbol de directorios que se muestra al hacer un nuevo *commit*. Esta prueba no ha reportado ningún error.

- Provocar que haya conflictos a la hora de combinar varias ramas. Si dos usuarios han modificado las mismas líneas de un fichero en sus copias locales y tratan de hacer un *merge* a una rama, lo que ocurre es que el primero que lo hace no tendrá problemas y el sistema almacenará los cambios realizados por éste. Pero cuando el segundo intente combinar sus cambios la aplicación detecta que la nueva copia está intentando realizar cambios sobre las mismas líneas de un fichero que han sido modificadas en otra versión anterior. Esto es lo que produce un conflicto, provocando que el sistema de control de versiones no pueda resolver de forma automática el *merge*.

  Para realizarlo manualmente, la aplicación muestra el fichero en conflicto y le da al usuario la opción de escoger los cambios que quiere que se almacenen, pudiendo escoger entre cualquiera de las dos versiones que entraron en conflicto. Tras seleccionar una de las dos versiones del fichero se realiza el *commit* con los nuevos cambios.

- Las tareas que necesitan interactuar con Git pueden provocar que el sistema tenga una respuesta lenta, y que dichas tareas tengan una duración que se debe tener en cuenta. Al realizar pruebas en la creación de *commits* y de proyectos, subiendo un fichero comprimido con un tamaño grande, el sistema generaba un error de *timeout*. Este error se produce cuando una acción tiene una duración mayor que el límite configurado en el servidor. Para solucionar ésto, sin cambiar la configuración del servidor, se puede emplear la función "set_time_limit" de PHP, en la que se indica el límite para este temporizador. Además de esta medida, también se depuraron algunas funciones para conseguir disminuir el tiempo de ejecución de las tareas.

- Cuando se descarga una versión almacenada en un repositorio se generaban problemas con el nombre del fichero comprimido que se creaba. Dependiendo del navegador con el que se ejecutaba, el nombre podía tener tildes o no. Estos fallos se producían en el navegador "Google Chrome". Para solucionar ésto, el sistema elimina las tildes que hay en el nombre del fichero que se genera en la descarga.

- Cancelar la realización de un *commit* cuando se ha enviado el fichero al servidor. La aplicación realiza correctamente esta acción, restaurando el

estado que tenía el directorio de trabajo de la rama cuando se empezó a realizar el *commit*.

Todas las pruebas realizadas sobre la aplicación han servido para solucionar una serie de problemas y también para realizar mejoras en la aplicación. Tras los cambios, se han vuelto a pasar las pruebas sobre la aplicación que finalmente ha pasado todas las pruebas sin generar errores, por lo que se puede decir que se ha conseguido la primera versión estable de la aplicación combinando todos los módulos.

# Capítulo 8

# Historia del proyecto

El primer paso en la realización del proyecto fue la planificación del mismo. Partiendo de los objetivos definidos inicialmente, se consultó la documentación de las diferentes tecnologías con las que se podía desarrollar una aplicación que los cumpliese. Tras conseguir toda la documentación, comenzó otra tarea en la que se buscaron aplicaciones cuya funcionalidad fuese similar a la de este proyecto, y sobre las que se realizaron algunas pruebas. Gracias a estas pruebas se sacaron ideas que se llevaron a la práctica a la hora de desarrollar la aplicación.

Al termino de todas estas tareas, y en base a la documentación recogida, se fijaron los requisitos de la aplicación.

La figura 8.1 presenta las tareas en las que se organizó la planificación, también se puede ver el *diagrama de Gantt* de la misma en la figura 8.2.

| # | Duración | Tarea | Inicio | Terminado | Predecesores |
|---|---|---|---|---|---|
| 1 | 13 días | **Análisis de requisitos** | 01/11/2010 | 17/11/2010 | |
| 2 | 3 días | **Definición de objetivos** | 01/11/2010 | 03/11/2010 | |
| 3 | 8 días | **Documentación** | 04/11/2010 | 15/11/2010 | 2 |
| 4 | 8 días | Control de versiones | 04/11/2010 | 15/11/2010 | |
| 5 | 8 días | Lenguajes de programación y frameworks | 04/11/2010 | 15/11/2010 | |
| 6 | 8 días | Búsqueda de aplicaciones similares | 04/11/2010 | 15/11/2010 | |
| 7 | 2 días | **Requisitos de la aplicación** | 16/11/2010 | 17/11/2010 | 3 |
| 8 | 2 días | Requisitos básicos | 16/11/2010 | 17/11/2010 | |
| 9 | 1 día | Requisitos opcionales | 16/11/2010 | 16/11/2010 | |

Figura 8.1: Tareas de la planificiación.

Tras el período de planificación, en el que se fijaron los objetivos, se consultó toda la documentación y se fijaron los requisitos del proyecto, comenzaron las tareas de desarrollo del mismo.

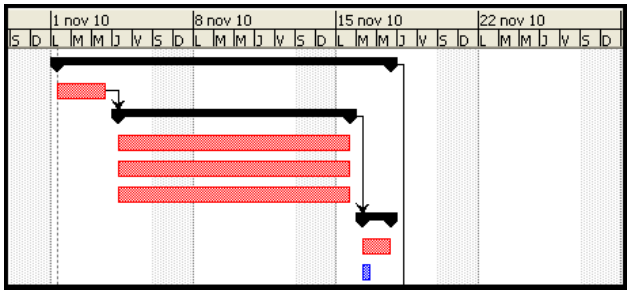


Figura 8.2: Diagrama de Gantt de la planificación.

El período de desarrollo está divido en diversas tareas, como se muestra en la figura 8.3, agrupadas en cuatro principales:

1. Diseño de la arquitectura y de los módulos que se deben implementar.
2. Desarrollo de la aplicación, con la interacción entre el sistema de control de versiones, la base de datos y las vistas de la aplicación Web.
3. Pruebas sobre la aplicación para comprobar el correcto funcionamiento.
4. Puesta en producción: una vez pasadas las pruebas, la aplicación debe ser publicada y comprobar el correcto acceso desde diversos navegadores, quedando publicada y lista para que los usuarios comiencen a hacer uso de la misma.

| # | Duración | Tarea | Inicio | Terminado | Predecesores |
|---|---|---|---|---|---|
| 1 | 62 días | **Desarrollo del proyecto** | 18/11/2010 | 11/02/2011 | |
| 2 | 7 días | **Diseño** | 18/11/2010 | 26/11/2010 | |
| 3 | 5 días | Arquitectura de la aplicación | 18/11/2010 | 24/11/2010 | |
| 4 | 2 días | Estructura de los contenidos | 25/11/2010 | 26/11/2010 | 3 |
| | | | | | |
| 5 | 43 días | **Desarrollo** | 29/11/2010 | 26/01/2011 | 4 |
| 6 | 18 días | **Interacción con GIT** | 29/11/2010 | 22/12/2010 | |
| 7 | 10 días | Creación de repositorios | 29/11/2010 | 10/12/2010 | |
| 8 | 5 días | Creación de versiones | 13/12/2010 | 17/12/2010 | 7 |
| 9 | 3 días | Creación de ramas | 20/12/2010 | 22/12/2010 | 8 |
| 10 | 25 días | **Lógica de la aplicación** | 23/12/2010 | 26/01/2011 | |
| 11 | 10 días | Interacción entre vistas y bbdd | 23/12/2010 | 05/01/2011 | 6 |
| 12 | 10 días | Maquetación de las vistas | 06/01/2011 | 19/01/2011 | 11 |
| 13 | 5 días | Cambios en el CSS | 20/01/2011 | 26/01/2011 | 12 |
| | | | | | |
| 14 | 9 días | **Pruebas en la aplicación** | 27/01/2011 | 08/02/2011 | 9 |
| 15 | 4 días | Interacción entre Git y PHP | 27/01/2011 | 01/02/2011 | |
| 16 | 2 días | Pruebas gestión repositorios | 02/02/2011 | 03/02/2011 | 15 |
| 17 | 3 días | Pruebas interacción página Web | 04/02/2011 | 08/02/2011 | 16 |
| | | | | | |
| 18 | 3 días | **Puesta en producción** | 09/02/2011 | 11/02/2011 | 17 |

Figura 8.3: Tareas implicadas en el desarrollo.

En la figura 8.4 se puede ver el diagrama de Gantt de las tareas en las que se estructuró el período de desarrollo.

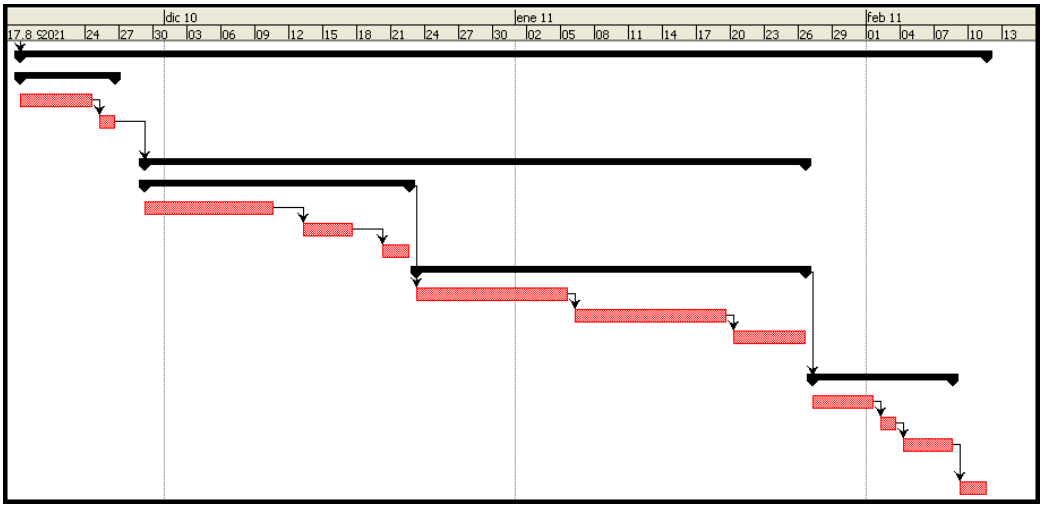


Figura 8.4: Diagrama de Gantt del desarrollo de la aplicación.

Una vez diseñada la arquitectura de la aplicación, se evalúan las tecnologías con las que se podría realizar dicha arquitectura, y se escoge de entre ellas aquellas con las que se trabajará finalmente. En el caso de este proyecto, la principal elección fue la de escoger PHP como el lenguaje de programación, y el sistema de control de versiones Git. Partiendo de estas dos tecnologías empezó el desarrollo del proyecto. En el desarrollo de aplicaciones Web intervienen varias líneas de trabajo, encargadas de la presentación de los contenidos en las diferentes vistas, y del desarrollo de la lógica de negocio requerida por la aplicación.

Los primeros pasos que se llevaron a cabo fueron intentos de comunicación entre el sistema de control de versiones y PHP. Una vez conseguida la comunicación entre las dos tecnologías, se empezó a trabajar en la aplicación para cumplir con los objetivos fijados. El primer hito que se alcanzó fue el de crear un repositorio a través de la interfaz Web, subiendo un fichero comprimido con el contenido del proyecto. Una vez conseguido ésto, se siguió con los siguientes objetivos que eran la creación de nuevos *commits* y de nuevas ramas en el repositorio previamente creado. Esta parte resultó más rápida gracias a los conocimientos adquiridos en la consecución del primer hito. A la vez que se trabaja en estos puntos, se iban estructurando los contenidos en las diferentes vistas.

Conseguida la comunicación entre Git y PHP, se realizaron el resto de las tareas, implicadas en la autenticación de usuarios, en el intercambio de mensajes, y se continuó con los trabajos de estructuración de los contenidos.

Una vez obtenida una versión estable de la aplicación, comenzaron las pruebas sobre ella. A medida que se iban pasando las pruebas si iban solucionando los errores que se encontraban, y se apuntaban posibles mejoras que se podrían llevar a cabo en el futuro. De todas las mejoras anotadas, algunas se implementaron ya que se consideró que dicha implementación no incrementaría los costes de tiempo en gran medida. Las que no se llevaron a cabo, pero que convendría estudiarlas en un futuro, se pueden ver en el capítulo 9.

## Capítulo 9

# Conclusiones y trabajos futuros

Tras el desarrollo de la aplicación se puede concluir que los objetivos fijados al inicio al del proyecto han sido cumplidos e incluso superados en algunos casos. A lo largo del período de desarrollo han ido surgiendo dificultades que se han ido superando hasta llegar al estado actual del proyecto.

## 9.1. Conclusiones.

Los objetivos fijados al inicio se centraban en la idea de conseguir interactuar con Git desde PHP para gestionar los repositorios que se almacenan en un servidor. Este objetivo inicialmente ya entraña su dificultad y la inversión de muchas horas de trabajo. A medida que se iba desarrollando la aplicación iban apareciendo nuevas ideas y necesidades. Estas ideas se separaban de los objetivos iniciales, pero se han intentado ir implementando, consiguiendo así dar mayor funcionalidad a la aplicación.

Tras finalizar el desarrollo del proyecto, implementando todos los módulos, se puede apreciar que la funcionalidad que tiene la aplicación en la versión actual es mucho mayor de lo que inicialmente se planteó. Ahora ya no sólo se interactúa con Git, también se controla el acceso de los usuarios, se pueden seleccionar varios idiomas para ver los contenidos, intercambiar mensajes entre los usuarios,etc.

Como conclusión final, decir que la aplicación Web desarrollada ha cumplido con todos los objetivos. Se ha conseguido desarrollar una aplicación que

permite trabajar con Git desde cualquier equipo sin tener que instalar ningún software adicional. Este era el principal objetivo del proyecto, y además se han añadido nuevas funciones a las que aparecían en el planteamiento inicial.

La versión actual estaría lista para desplegarse en un servidor y empezar a ponerla en producción. A pesar de acabar aquí el período de desarrollo de esta aplicación, hay que saber que aún no está terminada, es decir, se puede y se debe seguir evolucionando, añadiéndole nuevas mejoras y funciones que aún están por desarrollar.

## 9.2. Trabajos futuros.

Como todo proyecto, éste se puede y se debe seguir ampliando y mejorando. En este instante, el proyecto ha llegado a un punto en el que la funcionalidad que presenta es suficiente para poder fijar la primera versión estable del proyecto.

En el futuro se debería seguir trabajando en aspectos como la seguridad, el acceso concurrente y el retardo que presentan ciertas funciones. Se debe intentar reducir el tiempo de espera de los usuarios. Durante el desarrollo se han utilizado varios mecanismos que han reducido considerablemente el tiempo de espera.

Otro punto que se puede mejorar es el sistema de intercambio de mensajes. Ahora los usuarios pueden enviarse mensajes y comunicarse entre ellos sin problemas, pero considero que sería interesante incorporar un sistema de mensajería instantánea para darle mayor potencial a la comunicación entre los usuarios. También plantearía un nuevo modo para compartir comentarios en los *wikis* del proyecto. El actual cumple el objetivo de un *wiki*, pero se trata de una implementación básica, que a pesar de cumplir su función se podría cambiar por otra mucho más avanzada.

Y en cuanto a la funcionalidad que ofrece la aplicación respecto a la ofrecida por Git, se podría decir que se han conseguido implementar las principales funciones del sistema de control de versiones, pero aún faltan otras que se podrían añadir a la aplicación. Por tanto, se deberían seguir añadiendo y mejorando las funciones actuales, para conseguir implementar toda la funcionalidad que ofrece este sistema de control de versiones.

# Apéndice A

# Manual de instalación

A continuación se detallan las bibliotecas y las aplicaciones externas que se deben instalar en el servidor para que la aplicación Web funciones correctamente. La aplicación necesita que el siguiente software esté instalado en el servidor:

- Servidor Apache 2.2
- PHP versión 5.3.
- Mysql.
- *Framework* de Zend.
- Sistema de control de versiones Git.
- Software para trabajar con ficheros comprimidos.
- GeSHI.

## A.1. Instalación de servidor Apache.

El servidor web Apache se descarga desde la página Web de la Fundación Apache [6]. Se debe seleccionar el paquete, de la versión 2.2, en función de la plataforma en la que se vaya a instalar. En el caso de este proyecto se ha seleccionado la versión para Windows.

Al ejecutar el archivo descargado se lanza un asistente para la instalación, que irá informando de todos los pasos necesarios para completarla.

Realizando estas acciones quedaría instalado el servidor Apache, luego se pueden hacer configuraciones manuales editando el fichero "httpd.conf".

## A.2. Instalación de PHP.

La aplicación web ha sido programada en lenguaje PHP, por tanto será necesario tener corriendo un servidor que sea capaz de entender este lenguaje. La versión de PHP en la que ha sido desarrollado es la 5.3.1. En la página oficial de PHP existen manuales para configurar PHP en cada tipo de servidor [26].

Para instalarlo hay que ir a la página oficial de PHP y descargar el paquete con la versión indicada. Una vez descargado se debe extraer el contenido a una carpeta en el servidor. Después se copian todas las "dll" que se encuentran en la carpeta recién descomprimida a la carpeta "system32" dentro del directorio de instalación de Windows.

Una vez copiadas, se debe editar el fichero "php.ini", y se debe añadir al final del fichero de configuración de Apache los módulos nuevos instalados, para que éste los cargue al arrancar. Para ello se deben añadir las siguientes líneas al fichero "httpd.conf":

LoadModule php5_module "path_instalación_PHP/php5apache2_2.dll"

AddType application/x-httpd-php .php

PHPIniDir "path_instalación_PHP"

Estas líneas pueden varian en función de la versión descargada. El último paso, es incluir el directorio de instalación de PHP en la variable de entorno "PATH". Siguiendo estos pasos, y tras reiniciar el servidor, ya se podría comprobar el funcionamiento de aplicaciones desarrolladas con PHP.

## A.3. Instalación de MySQL.

También será necesario tener configurado en el servidor la base de datos que utiliza la aplicación. En el proyecto se ha utilizado la versión MySQL 5.1.41.

Para su instalación hay que ir a la página web oficial [14] y seleccionar el instalador que hay para la plataforma Windows. Este instalador guía al usuario en todos los pasos de la configuración.

Una vez realizados todos los pasos, queda instalada la base de datos en el servidor.

## A.4. Instalación del *framework* de Zend.

Para instalar el *framework* hay que obtener la última versión desde la página de Zend Framework [37]. La versión sobre la que se desarrollo la aplicación es la v1.11.2, que era la más actualizada a día 17 de Febrero de 2010.

Para realizar la instalación hay varias alternativas, en función de la plataforma sobre la que se instalará. Hay que seleccionar el paquete en la versión que se necesite, ya sea para Windows (fichero *zip*) o para Linux (fichero *tar*). Una vez copiado este paquete en el servidor hay que descomprimirlo en una carpeta.

Al descomprimir el fichero se crea una nueva estructura de ficheros. En ella aparece un directorio llamado "bin". En esta carpeta se encuentran los ejecutables del framework para los distintos sistemas operativas. Por tanto, para que nuestro servidor sepa donde se encuentran estos ficheros, debemos añadir la ruta hasta este directorio a la variable de entorno "PATH". Gracias a ésto se puede trabajar con el *framework*, pudiendo ejecutar comandos por línea de comandos para crear y modificar los controladores y las acciones actualmente desarrolladas.

## A.5. Instalación del sistema de control de versiones Git.

Para instalar Git hay que ir a la página oficial y descargarse la última versión. Para este proyecto se ha utilizado la versión v1.7.4. Existe un fichero ejecutable que lanza el instalador. El siguiente paso es indicar el directorio donde se quiere instalar, y luego seleccionar que no cambie los finales de linea de "CRLF" a "LF" para evitar problemas de compatibilidad.

Una vez que el instalador ha finalizado, ya está copiado todo lo necesario para trabajar con Git. El siguiente paso es configurar en PHP la ubicación donde se ha instalado Git. Para ésto hay que editar el fichero "php.ini" y añadir en la línea "include_path" la ruta absoluta hacia la carpeta "bin", que contiene el ejecutable del sistema de control de versiones. Esta carpeta se encuentra en el directorio donde se ha instalado Git.

Estos son los pasos necesarios para que la aplicación pueda interactuar con el sistema de control de versiones.

## A.6. Software para trabajar con ficheros comprimidos.

Para poder descomprimir ficheros en el servidor, es necesario utilizar dos programas, "unzip" y "unrar". Para poder trabajar con ellos, únicamente hay que descargar el fichero ejecutable desde el sitio Web de cada compañía. Una vez copiados en el servidor, hay que ubicar estos ficheros en una carpeta que esté incluida en la variable de entorno "PATH" de la plataforma, para que la aplicación desarrollada pueda localizarlos y poder interactuar con ellos.

## A.7. Instalación de GeSHI para mostrar el contenido de los ficheros

Para mostrar el contenido de los ficheros se utiliza una herramienta conocida como GeSHI [10]. Para instalarla, hay que ir a la página web del proyecto y descargar el fichero comprimido con el contenido del mismo. El contenido de este fichero se tiene que descomprimir en una carpeta llamada "geshi". Esta carpeta debe estar ubicada en el directorio "PEAR" que se encuentra dentro del directorio de instalación de PHP.

# Apéndice B

# Manual de usuario

## B.1. Trabajos sobre los repositorios

### B.1.1. Vista inicial

Esta es la vista a la que se accede después de autenticarse correctamente en la aplicación. El primer listado que aparece muestra todos los proyectos a los que el usuario puede acceder por ser el administrador, o por tener algún tipo de permiso de acceso al mismo.

En todas las vistas aparece un menú horizontal en la parte superior, que permitirá moverse por la aplicación de manera más rápida.


Figura B.1: Vista inicial de la aplicación.

### B.1.2. Listado de ramas creadas.

Esta es la vista a la que los usuarios acceden cuando seleccionan un proyecto en la vista inicial. En ella se listan todas las ramas, con información sobre el creador de la rama, la fecha de la última modificación y también una breve descripción del trabajo que se realiza en el proyecto. En todas las vistas se presenta también un menú con iconos que identifican cada una de las acciones que se pueden realizar sobre el repositorio desde cada vista.

En esta vista las tareas que se pueden realizar sobre el repositorio son:

1. Acceder al contenido de una rama del proyecto.
2. Acceder al *wiki* del proyecto, donde poder compartir comentarios con el resto de usuarios.
3. Realizar un *checkout* de una rama, es decir, crear una rama a partir de otra ya existente. Al seleccionar esta opción, aparecerá un campo de texto donde el usuario introducirá el nombre de la nueva rama. Esta rama tendrá el mismo historial que la rama origen, por lo que las dos cabeceras apuntarán a la misma versión.


Figura B.2: Vista con el listado de las ramas de un proyecto.

Si el usuario que accede a esta vista es el administrador del proyecto, usuario que lo ha creado, podrá realizar dos tareas de gestión adicionales:

1. Gestionar los permisos de acceso al proyecto.

2. Editar el fichero que contiene las reglas de los ficheros que el sistema de ignorar cuando se realizan los *commits*.

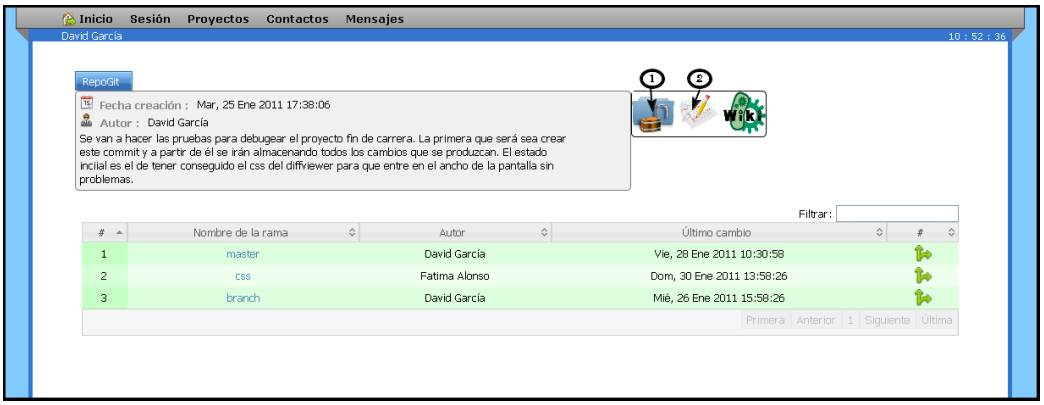


Figura B.3: Vista con el listado de ramas como administrador.

### B.1.3. Listado de las versiones almacenadas.

Cuando se selecciona una rama, se accede a esta vista donde se presenta una tabla con información de las diferentes versiones almacenadas en el resositorio. De cada una de las versiones se muestra el tiempo transcurrido desde que se generó el *commit*, el autor y la descripción de la misma.

Desde esta vista todos los usuarios pueden realizar una serie de acciones sobre los proyectos. Las principales acciones son:

1. Generar una nueva versión. Se muestra un formulario con los datos necesarios para poder realizar un nuevo *commit*.
2. Combinar la rama actual con la rama "master". Esto lo que hace es realizar un *merge* de la rama activa con la rama principal, volcando todos los cambios sobre ella. A la hora de realizar esta acción, se pueden provocar conflictos que deben ser solucionados manualmente. Esto se solucionará en la vista...
3. Acceder al *wiki* de la rama, para compartir comentarios específicos de la rama activa.
4. Descargar el contenido del proyecto en la versión seleccionada.

5. Realizar un *checkout* de la rama activa. El usuario indicará el nombre de la nueva rama.Con ésto se crea una nueva rama, con el nombre que indique el usuario. La cabecera de la nueva rama apuntará a la versión que se haya seleccionado.

6. Seleccionando el *hash* de una de las versiones se accede al listado de los ficheros incluidos en dicha versión.

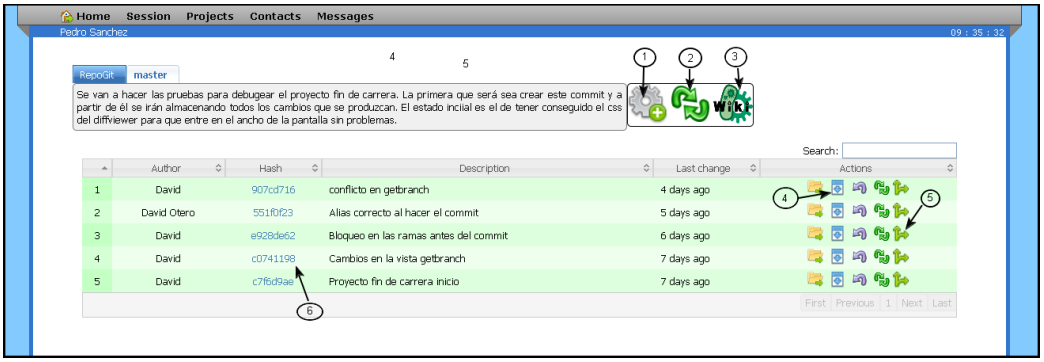


Figura B.4: Vista con el listado de versiones de un repositorio.

### B.1.4. Árbol de directorios de una versión.

Al seleccionar una versión, se accede al árbol de directorios con el contenido del proyecto en ese instante. En esta vista se muestra una breve descripción con datos de la versión, el árbol de directorios, y un menú con las opciones que se pueden realizar. Estas acciones son las siguientes:

1. Ver el árbol de directorios con todos los ficheros que contiene la versión actual.

2. Mostrar en el árbol de directorios únicamente los ficheros que han sido modificados en la versión actual.

3. Acceder al contenido de un fichero, haciendo *click* sobre su nombre.

4. Descargar el contenido de la versión seleccionada.

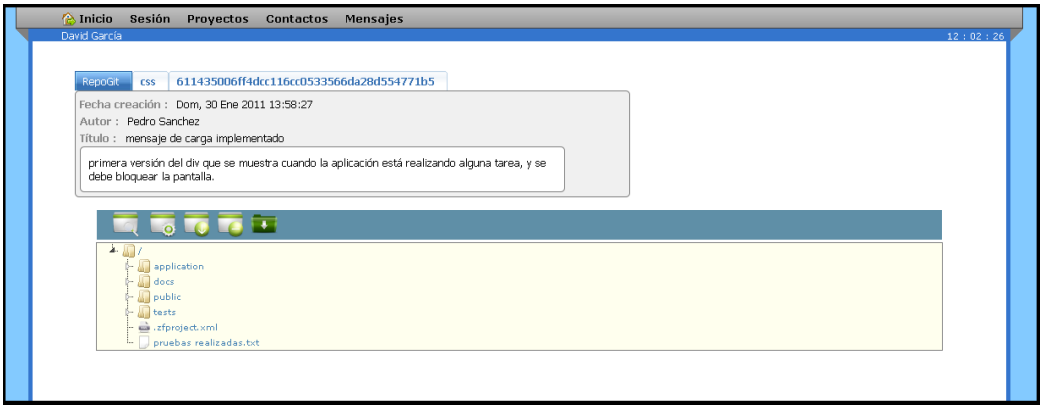


Figura B.5: Vista con el listado de los directorios y ficheros de una versión.

### B.1.5. Contenido de un fichero

Desde el árbol de directorios, mostrado en la figura B.5, se puede seleccionar un fichero para acceder a su contenido. Además de mostrar el contenido del ficheros, desde esta vista se puede acceder a la vista que presenta las diferencias entre el fichero y otra versión anterior.

La aplicación utiliza un *plugin* que muestra el texto con determinados colores en función de la extensión del fichero.

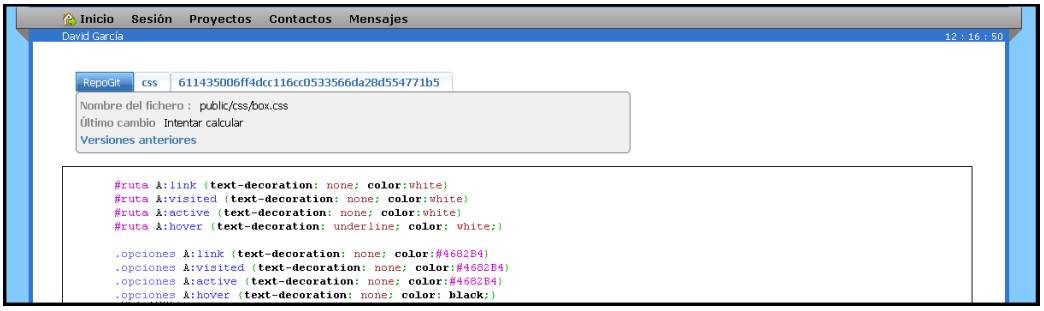


Figura B.6: Vista con el contenido de un fichero.

### B.1.6. Diferencias de un fichero entre dos versiones.

Esta vista muestra las diferencias entre dos versiones de un fichero. El usuario puede escoger la versión con la que se comparará el contenido del fichero en el estado activo. En la parte superior se muestra también un breve resumen con la información del fichero, y una lista desplegable con todas las versiones que tienen alguna modificación respecto a la versión activa del fichero. Los usuarios pueden escoger entre cualquiera de estas versiones para ver las diferencias existentes.

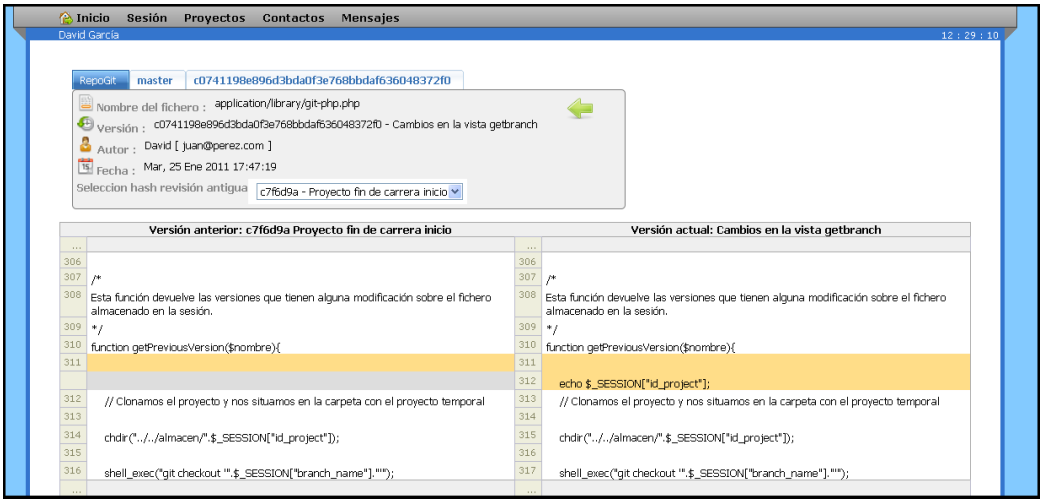


Figura B.7: Vista con las diferencias en un fichero entre dos versiones.

### B.1.7. Creación de nuevos proyectos.

Para crear un nuevo proyecto lo único que se necesita es el nombre del mismo y un fichero comprimido, con extensión ZIP, que contenga los ficheros del nuevo proyecto. En el servidor se recogen las nuevas peticiones y la aplicación WEB realiza las siguientes acciones:

1. Se inserta en la base de datos una nueva entrada con los datos necesarios del nuevo proyecto: título, nombre e identificador del autor y la fecha de creación.

2. Mediante una llamada a Git, por línea de comandos, se inicia el nuevo repositorio. Para identificar al repositorio se utiliza el "id" de la fila

insertada previamente.

3. Una vez iniciado el nuevo repositorio, se descomprime en él todo el contenido del fichero comprimido que se recibe en la petición *http*.

4. Se realiza el primer *commit*, añadiendo todos los ficheros nuevos que se han descomprimido previamente en el repositorio. Esto se realiza con una llamada por línea de comandos. Si el usuario a rellenado el campo con el texto para el primer *commit* se le indicará éste a Git, en caso contrario se coge uno por defecto.

Una vez realizados todos estos pasos, el nuevo repositorio queda creado y alojado en el servidor, y ya se puede empezar a interactuar con el proyecto.

### B.1.8. Ficheros ignorados al realizar *commit*.

El sistema de control de versiones se puede configurar para que tenga en cuenta las reglas incluidas en un fichero para que sean ignoradas a la hora de hacer *commit*. Si Git no debe controlar los cambios sobre ficheros de una determinada extensión, o con un determinado nombre, se debe acceder a esta vista e incluir las reglas que se deban tener en cuenta.

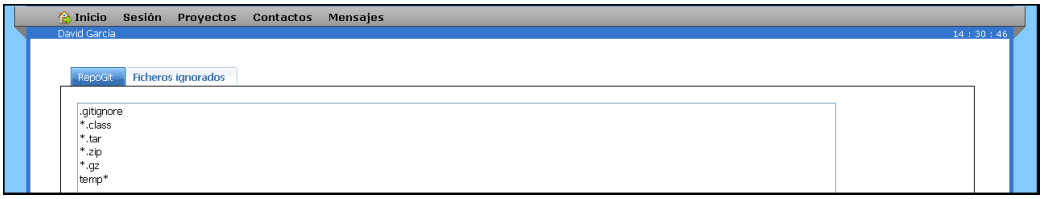


Figura B.8: Ficheros ignorados en los commits.

### B.1.9. Mantenimiento de las versiones.

Para almacenar nuevas versiones en un proyecto el usuario debe completar una serie de pasos:

1. Seleccionar un fichero comprimido con los ficheros que van a incluirse en la nueva versión. Cuando se selecciona este fichero, se realiza una

petición *ajax* asíncrona al servidor mediante *jQuery*. El servidor al recibir esta petición clona el proyecto y descomprime el contenido del fichero subido en el directorio de trabajo del proyecto clonado.

2. Tras rellenar el formulario con los datos de la nueva versión: título, descripción y seleccionar si se quieren buscar ficheros eliminados en la nueva versión se envía el formulario al servidor.

3. Se consulta el estado del repositorio, para ello se hace una llamada por línea de comandos y se parsea la respuesta que se obtiene. Con esto se genera un array con el listado de todos los ficheros añadidos, modificados y eliminados, estos últimos si se ha seleccionado la opción de controlar los ficheros eliminados.

4. Se presenta el listado de ficheros distinguiendo el estado cada uno y dando la opción, mediante un *checkbox*, de seleccionar todos los cambios que se van a incluir en la nueva versión.

## B.2. Usuarios e intercambios de mensajes.

### B.2.1. Listado con los usuarios registrados

Muestra información de todos los contactos registrados en la aplicación, indicando su nombre, dirección de correo y mostrando su imagen si el usuario la ha subido préviamente.

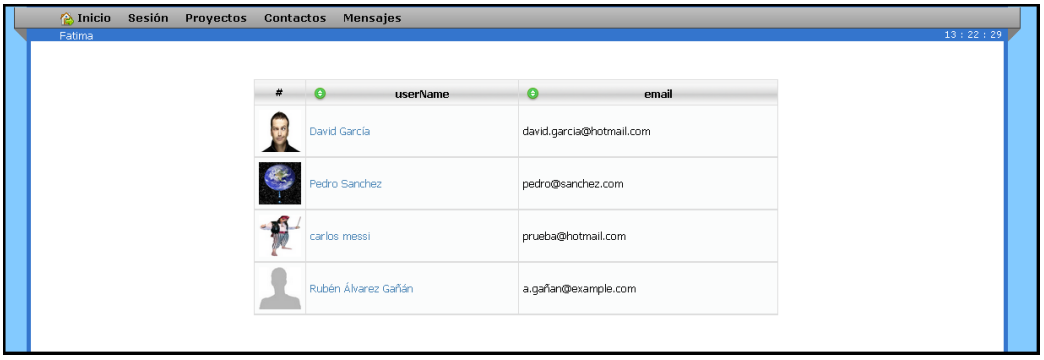


Figura B.9: Listado de los usuarios registrados.

### B.2.2. Listado con los grupos creados.

Esta vista muestra el listado de los grupos creados organizados en grupos creados por el usuario que está utilizando la aplicación, grupos públicos, y otra con el listado de los grupos de los que ya es miembro el usuario.

Desde esta vista los usuarios pueden crear nuevos grupos, unirse o abandonar un grupo ya existente, y acceder al área de administración de los grupos que hayan creado.


Figura B.10: Listado de grupos registrados en la aplicación.

### B.2.3. Buzón de mensajes.

En esta vista se accede a un listado con los mensajes que el usuario ha recibido. Desde esta vista se puede responder o eliminar los mensajes recibidos.

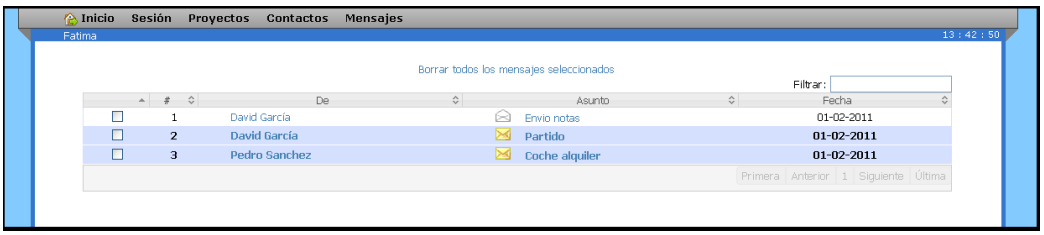


Figura B.11: Listado de mensajes recibidos.

## B.3. Permisos de acceso.

Esta vista presenta el listado de los usuarios que tiene permisos para acceder al repositorio y realizar cambios en el mismo. Un usuario puede acceder al repositorio por tener permisos él mismo, o por pertenecer a un grupo que tenga permisos de acceso. Desde esta vista se podrá permitir o bloquear el acceso a los usuarios, y agregar o eliminar grupos al listado de grupos permitidos.

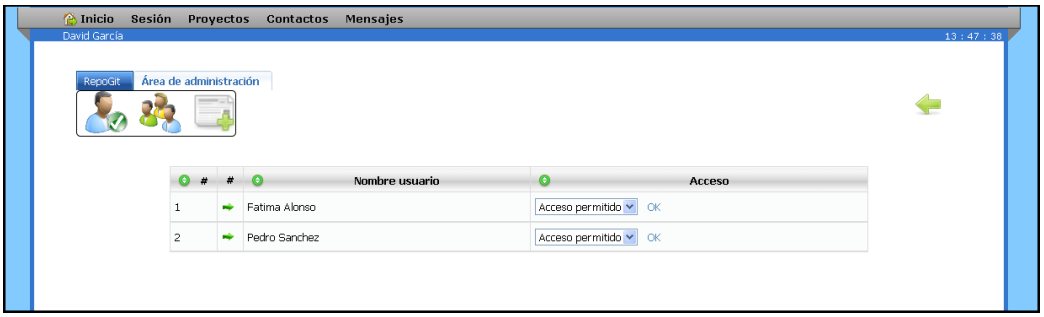


Figura B.12: Permisos de acceso al repositorio.

## Apéndice C

# Presupuesto

# UNIVERSIDAD CARLOS III DE MADRID
## Escuela Politécnica Superior

**PRESUPUESTO DE PROYECTO**

1.- Autor: David Otero Gutiérrez

2.- Departamento: Telemática

3.- Descripción del Proyecto:

- Titulo **Aplicación Web para control de versiones de software.**
- Duración (meses) **6 meses**
- Tasa de costes Indirectos: **20%**

4.- Presupuesto total del Proyecto:

Euros 35.000,00 €

5.- Desglose presupuestario (costes directos)

**PERSONAL**

| Apellidos y nombre | N.I.F. | Categoría | Dedicación meses | Coste hombre mes (Euro) | Coste Total(Euro) |
|---|---|---|---|---|---|
| Otero Gutiérrez, David | xxxxxxxx | Ingeniero | 6 | 3.500,00 | 21.000,00 |
| | | | | **Total** | 21.000,00 |

**EQUIPOS**

| Descripción | Coste (Euro) | % Uso dedicado proyecto | Dedicación (meses) | Periodo de depreciación | Coste imputable |
|---|---|---|---|---|---|
| Ordenador para desarrollar | 1.500,00 | 100 | 6 | 20 | 450,00 |
| Licencia de windows | 300,00 | 100 | 6 | 60 | 30,00 |
| | | | | **Total** | 480,00 |

**SUBCONTRATACIÓN DE TAREAS**

| Descripción | Empresa | Coste imputable |
|---|---|---|
| Consultas tecnologías | Jesús Arias Fisteus | 1.300,00 |
| Pruebas aplicación | Fátima de la Osa | 1.500,00 |
| | **Total** | 2.800,00 |

6.- Resumen de costes

| Presupuesto | **Costes Totales** |
|---|---|
| Personal | 21.000 |
| Amortización | 480 |
| Subcontratación de tareas | 2.800 |
| Costes Indirectos | 4.856 |
| Total | 29.136 |

# Bibliografía


[1] The common gateway interface (cgi) version 1.1. http://www.rfc-editor.org/rfc/rfc3875.txt, Octubre 2004.

[2] Lenguaje de programacion para web:asp. http://www.scribd.com/doc/2413026/Lenguaje-de-programacion-para-Web-ASP, Marzo 2008.

[3] 10 useful php framework from web. http://www.dreamcss.com/2009/06/10-useful-php-framework-from-web.html, Junio 2009.

[4] Glip (git library in php) - api documentation. http://fimml.at/glip/doxygen/, Mayo 2009.

[5] Akelos php framework. http://www.akelos.org/, Octubre 2010.

[6] Apache. http server project. http://httpd.apache.org/, Noviembre 2010.

[7] Cakephp - framework para php. http://cakephp.org/, Octubre 2010.

[8] Codeigniter - framework para php. http://codeigniter.com/, Octubre 2010.

[9] Control de versiones con git y github. http://www.slideshare.net/guest638090/control-de-versiones-con-git-y-github, Junio 2010.

[10] Geshi - generic syntax highlighter. http://qbnz.com/highlighter/, Diciembre 2010.

[11] jquery: Write less.do more. http://jquery.com/, Marzo 2010.

[12] jstree - jquery tree plugin. http://www.jstree.com/, Noviembre 2010.

[13] Lenguaje de programacion ruby. http://www.ruby-lang.org/es/, Marzo 2010.

[14] Mysql 5.0 reference manual. http://dev.mysql.com/doc/refman/5.0/es/index.html, Marzo 2010.

[15] Pclzip - php compression library. http://www.phpconcept.net/pclzip/, Febrero 2010.

[16] Php: Hypertext preprocessor. http://www.php.net/, Marzo 2010.

[17] Php on trax - rapid php application development. http://www.phpontrax.com/, Octubre 2010.

[18] Python programming language - official website. http://www.python.org/, Septiembre 2010.

[19] Python programming language -official website. http://www.python.org/, Septiembre 2010.

[20] Wikipedia - características prinicipales framework. http://es.wikipedia.org/wiki/Framework, Octubre 2010.

[21] Wikipedia - comparison of open source software hosting facilities. http://en.wikipedia.org/wiki/Comparison_of_open_source_software_hosting_facilities, Noviembre 2010.

[22] Wikipedia - motor zend. http://es.wikipedia.org/wiki/Motor_Zend, Mayo 2010.

[23] Git - fast version control system. http://git-scm.com/, Enero 2011.

[24] Historia de php. http://php.net/manual/es/history.php.php, Febrero 2011.

[25] The perl programming language. http://www.perl.org/, Enero 2011.

[26] Php: Installation and configuration. http://www.php.net/manual/en/install.php, Febrero 2011.

[27] Wikipedia - applet java. http://es.wikipedia.org/wiki/Applet_Java, Febrero 2011.

[28] Wikipedia - lenguaje interpretado vbscript. http://es.wikipedia.org/wiki/VBScript, Febrero 2011.

[29] World wide web consortium (w3c) homepage. http://www.w3.org/, Febrero 2011.

[30] Javier Egiluz. Symfony - framework para php. http://www.symfony.es/, Octubre 2010.

[31] Chas Emerick. A javascript visual diff tool & library. http://snowtide.com/jsdifflib, Febrero 2007.

[32] Apache Software Foundation. Subversion - version control system. http://subversion.apache.org/, Octubre 2010.

[33] Prado Group. Component framework for php 5. http://www.pradosoft.com/, Febrero 2011.

[34] Allan Jardine. jquery plugin datatables. http://www.datatables.net/, Febrero 2011.

[35] Rails. Ruby on rails - open source web framework. http://rubyonrails.org/, Febrero 2011.

[36] Kohana team. Kohana - framework para php. http://kohanaframework.org/, Octubre 2010.

[37] Zend Technologies. Zend framework homepage. http://framework.zend.com/, Febrero 2011.

[38] W3C. Html 4.01 specification. http://www.w3.org/TR/REC-html40/, Diciembre 1999.

[39] W3C. Guia breve de css. http://www.w3c.es/divulgacion/guiasbreves/hojasestilo, Noviembre 2010.

[40] W3C. Html5 - specification. http://www.w3.org/TR/html5/, Enero 2011.

[41] Ronnie Garcia y Travis Nickels. Plugin jquery uploadfy - envío múltiple de ficheros. http://www.uploadify.com/about/, Febrero 2011.